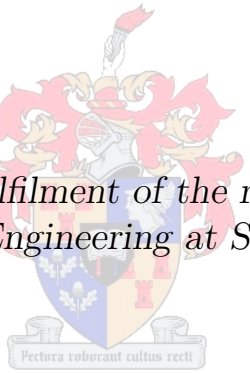
Markerless Augmented Reality on Ubiquitous Mobile Devices with Integrated Sensors

BY

Carel van Wyk



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at Stellenbosch University

Supervisor: Dr. H.A. Engelbrecht

Department of Electrical & Electronic Engineering

March 2011

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2011

Opsomming

Sleutelwoorde: Toegevoegde Realiteit, rekenaarvisie, selfoon, geïntegreerde sensor, iPhone

Die berekeningskrag van nuwe-generasie selfone neem elke dag toe en kragtige "slim-fone" word al hoe meer populêr onder verbruikers. Die tegniese spesifikasies van 'n nuwe slim-foon vandag is vergelykbaar met dié van 'n persoonlike rekenaar van slegs 'n paar jaar gelede. Die kombinasie van kragtige verwerkers, kameras en die gemaklikheid waarmee programmatuur op hierdie toestelle ontwikkel word, maak dit 'n aantreklike ontwikkelingsplatform vir navorsers in Toegevoegde Realiteit.

Die implimentering van 'n merker-gebaseerde Toegevoegde Realiteitstelsel op selfone is 'n probleem wat reeds grotendeels opgelos is. Merker-vrye stelsels, aan die ander kant, bied steeds interessante uitdagings omdat hulle meer prosesseringskrag vereis. 'n Paar navorsers het reeds rekenaarvisie-gebaseerde merker-vrye stelsels aangepas om op selfone te funksioneer. In hierdie tesis stel ons die ontwikkeling voor van 'n hibriede stelsel wat gebruik maak van rekenaarvisie sowel as geïntegreerde sensore in die foon om die berekening van kamera-orientasie te vergemaklik.

Ons gebruik geïntegreerde sensore om drie uit ses vryheidsgrade van orientasie te bereken, terwyl die oorblywende drie met behulp van rekenaarvisie-tegnieke bepaal word. 'n Prototipe stelsel is ontwikkel as deel van hierdie tesis.

Abstract

Keywords: Augmented Reality, AR, markerless, mobile, hybrid, computer vision, sensor fusion, iPhone.

The computational power of mobile smart-phone devices are ever increasing and high-end phones become more popular amongst consumers every day. The technical specifications of a high-end smart-phone today rivals those of a home computer system of only a few years ago. Powerful processors, combined with cameras and ease of development encourage an increasing number of Augmented Reality (AR) researchers to adopt mobile smart-phones as AR platform.

Implementation of marker-based Augmented Reality systems on mobile phones is mostly a solved problem. Markerless systems still offer challenges due to increased processing requirements. Some researchers adopt purely computer vision based markerless tracking methods to estimate camera pose on mobile devices. In this thesis we propose the use of a hybrid system that employs both computer vision and integrated sensors present in most new smart-phones to facilitate pose estimation.

We estimate three of the six degrees of freedom of pose using integrated sensors and estimate the remaining three using feature tracking. A proof of concept hybrid system is implemented as part of this thesis.

Acknowledgements

I would like to express my sincere gratitude towards the following people:

- My promoter, Dr. H.A. Engelbrecht for his guidance, insight and recommendations.
- Dr. Gert-Jan van Rooyen for his valued inputs.
- My fellow Medialab colleagues.
- MIH, Antonie Roux and Jacques Van Niekerk for making this research possible and providing life-changing opportunities.
- My family and friends, specifically my parents, Jessica and Dirk for their support.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

2-D	Two Dimensional
3-D	Three Dimensional
3-DOF	Three Degrees of Freedom
6-DOF	Six Degrees of Freedom
AR	Augmented Reality
CAD	Computer Aided Design
CPU	Central Processing Unit
CV	Computer Vision
DOF	Degrees of Freedom
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FPS	Frames Per Second
GLONASS	Global Navigation Satellite System
GPS	Global Position System
GPU	Graphics Processing Unit
HMD	Head Mounted Display
Hz	Hertz
ISMAR	International Symposium on Mixed and Augmented Reality
LED	Light-Emitting Diode
LCD	Liquid Crystal Display
LMedS	Least Median of Squares
MB	Megabyte
MEMS	Micro-Electro-Mechanical
ms	Milliseconds
MIT	Massachusetts Institute of Technology
OLED	Organic Light-Emitting Diode
PC	Personal Computer
PTAM	Parallel Tracking and Mapping
RAM	Random Access Memory
RANSAC	RANdom SAmple Consensus
RAPiD	Real-time Attitude and Position Determination

SLAM	Simultaneous Localisation and Mapping
SIFT	Scale Invariant Feature Transform
SSD	Sum of Squared Differences
SURF	Speeded-Up Robust Features
SUSAN	Smallest Univalue Segment Assimilating Nucleus
USURF	Upright Speeded-Up Robust Features
UI	User Interface
VR	Virtual Reality

Variables

symbol	**description**
κ	Empirically determined weighting factor ranging from 0.04 to 0.15
λ_1	First eigenvalue of $\mathbf{H}_{Harris}$
λ_2	Second eigenvalue of $\mathbf{H}_{Harris}$
Π_l	Left image plane
Π_r	Right image plane
$\vec{a}$	Accelerometer's instantaneous acceleration vector
$\vec{a}_i$	i-th instantaneous acceleration vector
$\bar{a}$	Accelerometer's steady-state acceleration vector
$\bar{a}_i$	i-th steady-state acceleration vector
$\bar{a}_x$	x-axis component of $\bar{a}$
$\bar{a}_y$	y-axis component of $\bar{a}$
$\bar{a}_z$	z-axis component of $\bar{a}$
$\hat{a}$	Unit vector form of $\bar{a}$
$\|\|\bar{a}\|\|$	Length of $\bar{a}$
$\mathbf{A}$	A helper matrix used in calculation of optical flow
$\vec{b}$	An arbitrary vector perpendicular to $\bar{a}$
$\|\|\vec{b}\|\|$	Length of $\vec{b}$
$\hat{b}$	Unit vector form of $\vec{b}$
$B(x, y)$	Blue value of a two-dimensional colour image at position (x, y)
$\vec{c}$	A helper vector used in calculation of optical flow
c_x	Imager x-axis offset relative to centre of projection due to sensor imperfections
c_y	Imager y-axis offset relative to centre of projection due to sensor imperfections
C	Interest point response function
C_{Harris}	Harris corner detector interest point response function
$C_{Shitomasi}$	Shi-Tomasi corner detector interest point response function
$\vec{d}$	A helper vector used in calculation of optical flow
dx	Delta-x movement of a point on a two-dimensional image.

symbol	**description**
dy	Delta-y movement of a point on a two-dimensional image.
$\sum dx$	Sum of Haar wavelet responses along the x- axis.
$\sum dy$	Sum of Haar wavelet responses along the y- axis.
$\sum \lvert dx \rvert$	Sum of the absolute values of Haar wavelet responses along the x- axis.
$\sum \lvert dy \rvert$	Sum of the absolute values of Haar wavelet responses along the y- axis.
e_l	Epipole on left image plane
e_r	Epipole on right image plane
$\mathbf{E}$	The Essential Matrix
$f(x, i)$	Intensity of a point at x at time i
f	Focal length of a camera
f_x	Effect of rectangular pixels on focal length along the x-axis
f_y	Effect of rectangular pixels on focal length along the y-axis
$\mathbf{F}$	The Fundamental Matrix
g	Unit of gravitational acceleration on earth. 1 $g = 9.8\ m/s^2$
$G(x, y)$	Green value of a two-dimensional colour image at position (x, y)
h	Projected height
h'	Projected height onto an image plane
H	Height
$\mathbf{H}$	3-by-3 Homography matrix
$\mathbf{H}_{Harris}$	Harris Matrix
i	A point in time
di	A time delta
$I(x, t)$	Image intensity of a one-dimensional image at position x at time t
$I(x, y)$	Image intensity of a two-dimensional image at position (x, y)
$I(x_p, y_p)$	Image intensity of a two-dimensional image at p
I	A two-dimensional greyscale image
$I_{\sum}$	The integral image of I
I_x	First-order partial derivative of I in the x-direction
I_y	First-order partial derivative of I in the y-direction
I_i	First-order partial derivative of I over time, i
$\mathbf{M}$	Camera intrinsics matrix
$\mathbf{M}_l$	Intrinsic parameters of left imager
$\mathbf{M}_r$	Intrinsic parameters of right imager
$\mathbf{M}_{modelview}$	OpenGL Model-view Matrix
$\vec{n}$	Normal vector on the plane defined by P, O_l and O_r
O_l	Left centre of projection
O_r	Right centre of projection
p	An arbitrary pixel in a two-dimensional image
p'	An arbitrary pixel in a one-dimensional image

symbol	**description**
$\vec{p_l}$	Projection of P on left image plane
$\vec{p_r}$	Projection of P on right image plane
P	A point in 3-D real-world coordinates
$\vec{P_l}$	The location of P relative to O_l in 3-D real-world coordinates
$\vec{P_r}$	The location of P relative to O_r in 3-D real-world coordinates
q	A projection of a real-world point onto a 2-D plane
$\tilde{q}$	q in homogeneous coordinate form
$\tilde{q}'$	$\tilde{q}$ with its component $\tilde{w}$ restricted to 1
Q	A point in 3-D real-world coordinates
$Q1$	Array of points in quadrant one (top left) of an image
$Q2$	Array of points in quadrant two (top right) of an image
$Q3$	Array of points in quadrant three (bottom left) of an image
$Q4$	Array of points in quadrant four (bottom right) of an image
Q_1	Specific points in 3-D real-world coordinates
Q_2	Specific points in 3-D real-world coordinates
$\tilde{Q}$	Q in homogeneous coordinate form
$\tilde{Q}'$	$\tilde{Q}$ with its component Z restricted to 0
$\tilde{Q}''$	A form of $\tilde{Q}'$ with the Z component left out
$\vec{r}_1$	The first 3-by-1 column vector of $\mathbf{R}$
$\vec{r}_2$	The second 3-by-1 column vector of $\mathbf{R}$
$\vec{r}_3$	The third 3-by-1 column vector of $\mathbf{R}$
$\mathbf{R}$	Rotation matrix.
$\tilde{\mathbf{R}}$	$\mathbf{R}$ in homogeneous form.
$R(x, y)$	Red value of a two-dimensional colour image at position (x, y)
s	Scale factor
S	Sum of squared differences of a shifted patch around an original patch
S_{bright}	Set of contiguous pixels around p brighter than p by t
S_{dark}	Set of contiguous pixels around p darker than p by t
$S(x, y)$	Sum of squared differences over an area around (x, y)
$\mathbf{S}$	Matrix representation of a cross product with T
t	Corner detection threshold
$\vec{t}$	Translation vector consisting of three of the six DOF making up pose.
$\vec{t}_i$	ith translation vector
$\vec{t}_{image}$	The relative translation between two frames in image coordinates.
$\vec{T}$	The position of O_r relative to O_l in real-world coordinates
T_x	x-axis component of T
T_y	y-axis component of T
T_z	z-axis component of T
$T_{x,total}$	Total camera translation relative to the x-axis.

symbol	**description**
$T_{y,total}$	Total camera translation relative to the y-axis.
$T_{z,total}$	Total camera translation relative to the z-axis.
$\vec{t}_{Total}$	Total camera translation.
$\vec{t}_{Total,i}$	ith total camera translation.
u	An arbitrary pixel on the contiguous FAST circle around p
$\vec{u}$	Movement of a point on a 2-D image in the x-direction
$\vec{v}$	Movement of a point on a 2-D image in the y-direction
V	Corner detection candidate score function
$\tilde{w}$	Additional dimension variable used to form $\tilde{q}$
$w(u, v)$	Weighting factor for a patch position (u, v)
$\mathbf{W}$	Helper matrix, combination of $\mathbf{R}$ and $\vec{t}$
x	x-axis position of the projection of a real-world point onto a 2-D plane
x'	Rotated camera x-axis
x_{image}	Corrected x-axis position of a real-world point projected onto an image plane
$\tilde{x}$	x in homogeneous coordinate form
$\vec{x}''$	Instantaneous acceleration vector returned by x-axis of accelerometer
$\bar{x}'$	Steady-state x-axis acceleration vector
Δx	The x-axis difference in position between two frames in image coordinates
Δx_n	The x-axis difference in position between all points in quadrant "n"
X	X-axis position of a point in 3-D real-world coordinates
ΔX	The X-axis difference in position between two points in real-world coordinates
X_1 & X_2	Specific X-axis positions of points in 3-D real-world coordinates
y	y-axis position of the projection of a real-world point onto a 2-D plane
y'	Rotated camera y-axis
$\tilde{y}$	y in homogeneous coordinate form
y_{image}	Corrected y-axis position of a real-world point projected onto an image plane
$\vec{y}''$	Instantaneous acceleration vector returned by y-axis of accelerometer
$\bar{y}'$	Steady-state y-axis acceleration vector
Δy	The y-axis difference in position between two frames in image coordinates
Δy_n	The y-axis difference in position between all points in quadrant "n"
Y	Y-axis position of a point in 3-D real-world coordinates
ΔY	The Y-axis difference in position between two points in real-world coordinates
Y_1 & Y_2	Specific Y-axis positions of points in 3-D real-world coordinates
z'	Rotated camera z-axis
$\vec{z}''$	Instantaneous acceleration vector returned by z-axis of accelerometer
$\bar{z}'$	Steady-state z-axis acceleration vector
Δz	The z-axis difference in position between two frames in image coordinates
Δz_n	The z-axis difference in position between all points in quadrant "n"
Z	Z-axis position of a point in 3-D real-world coordinates

symbol	**description**
ΔZ	The Z-axis difference in position between two points in real-world coordinates
Z_1 & Z_2	Specific Z-axis positions of points in 3-D real-world coordinates

Chapter 1

Introduction

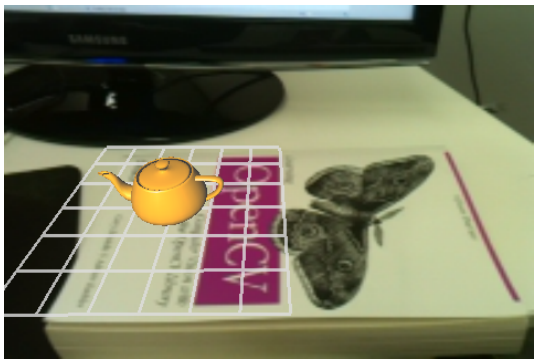
1.1 Augmented Reality

Augmented Reality (AR) is the combination of real and virtual environments. The term "Augmented Reality" was coined in 1990 by Tom Caudell while developing a system for Boeing to help workers assemble aircraft with the aid of screens that projected blueprint information on a worker's view. Application spaces for AR include entertainment, education, art, navigation, visualisation, manufacturing, medical and military fields. The processing power of home computers and mobile devices have steadily increased over the years to enable the use of more and more computationally intensive Computer Vision (CV) techniques. Every year has brought advances in the fields of computer vision and Augmented Reality in general. In recent years, the processing capabilities of mobile smart-phones have reached a stage where they have become a viable platform for consumer level Augmented Reality applications. This increase in processing power combined with integrated orientation sensors present in most new smart-phones [23] is making smart-phones an increasingly popular platform for Augmented Reality research.

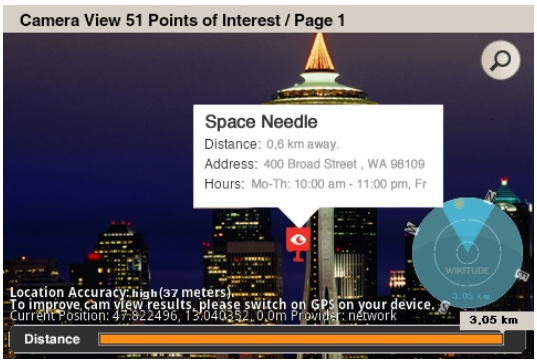
Augmented Reality is closely linked to Virtual Reality (VR). Where Virtual Reality is the immersion of a subject into a purely virtual world, Augmented Reality is the immersion of a subject into a version of the real world that is augmented with additional information. Virtual objects are overlaid on top of a real view of the real world in real-time and must be drawn in such a way as to appear a natural part of the real world.

Figure 1.1 demonstrates two examples of Augmented Reality. On the left, a virtual object (teapot) is drawn on top of a view of the real world. The virtual object is transformed to reflect any change in rotation or translation of the view so that it appears to be a natural part of the real world. Figure 1.1(a) was generated from the proof of concept AR application developed for this thesis. On the right is an example of a different application for AR, namely navigation. Objects in the real world are tagged with additional information and the information is superimposed on top of the view of real world objects.

Azuma [3] defines AR by stating that all AR systems must exhibit three characteristics. It must



(a) A virtual object (teapot) is drawn so that it appears to rest on top of a real world object (book).



(b) Additional information related to a real world object is attached to the object on the display. Picture courtesy of [28].

Figure 1.1: *Examples of Augmented Reality.*

1. combine the real and the virtual,
2. be interactive in real-time and
3. be registered in 3-D.

This definition avoids limiting AR to specific technologies, such as Head-Mounted Displays (HMD), while excluding concepts such as computer generated content in films (not real-time) and 2-D overlays such as "Heads-Up Displays" (not registered in 3-D).

1.2 Types of Augmented Reality Systems

In the past, AR applications were mostly limited to computers due to high processing requirements. We have seen increases in the processing capabilities of mobile devices every year and much research has already been done on Augmented Reality on mobile devices such as tablet PCs [21], laptops [26] and mobile phones [19].

In terms of visual tracking, there are three types of AR systems: marker-based, model-based and markerless trackers. Marker-based systems have been around for many years and several libraries have been developed that allows for the rapid development of marker-based applications on a wide range of devices including mobile phones. One of the most popular libraries is known as ARToolkit[1]. Figure 1.2 shows an example of a mobile marker-based AR application we developed in less than one week using the ARToolkit library.

Model-based systems can track 3-D objects of known shape. An example of a model-based system would be the tracking of a model aeroplane through the air or a system that can determine its position and orientation based on a 3-D map of its environment.

[1] `http://www.hitl.washington.edu/artoolkit/`

Figure 1.2: *ARToolkit based AR business card application.*

Markerless systems track points or *features* that have not been previously determined. "Points of interest" are identified and tracked in real-time and their movement analysed to determine the position and orientation (pose) of the observing camera. Once the pose of the camera is known, 3-D objects may be transformed and rendered to appear as though they are a part of the real world. Markerless tracking systems require more processing power than marker- or model-based approaches.

Existing marker-based, model-based and markerless AR systems are discussed in more detail in chapter 2.

1.3 Physical Components of an Augmented Reality System

An Augmented Reality system depends on four physical components:

- **Orientation and input sensors**: Provides the system with visual input, user input and potentially with data that can aid orientation.
- **Data Feed**: Provides information applicable to the environment around the AR system with which to augment the user's view.
- **User feedback peripherals**: Primarily in the form of visual output, but may also include audio feedback or other more exotic user interfaces.
- **Processing Unit**: Combines the data from input sensors to determine orientation and augments the user's view with information from the data feed. Outputs the result to the user feedback peripherals.

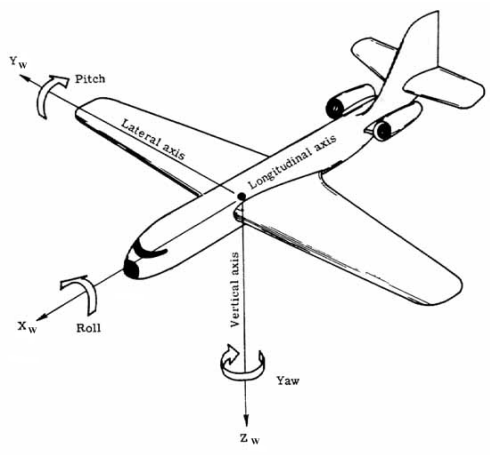


Figure 1.3: *Visualisation of the three angles of orienation: pitch, roll and yaw. Picture courtesy of NASA.*

1.3.1 Orientation and Input Sensors

An AR system must be aware of its orientation and position within the surrounding environment in order to augment the environment. The combination of an observer's orientation and position is known as *pose*. Pose will be defined in more detail in section 3.1. Pose may be determined by using computer vision, but there are alternative methods to estimate pose in the form of hardware sensors designed specifically to measure some or all of the six degrees of freedom parameters that make up pose.

Location

A popular way of determining one's position on earth is by utilising the Global Positioning System (GPS) developed by the United States military and became fully operational in 1994. It was available to civilians in degraded form until 2000 when "Selective Availability" was discontinued and it was made freely available world wide. Alternatives include Cell phone tower based location determination and Russia's GLObal Navigation Satellite System (GLONASS), but the former does not provide sufficient accuracy and the latter is not currently fully operational. GPS is sufficiently cheap and accurate for determining location in AR applications, and GPS sensors have been included in mobile phones since 2005. Wikitude[2] (2009) and Layar[3] (2009) are two popular mobile GPS-based Augmented Reality apps.

Orientation

In 3-D space, orientation of a rigid body can be uniquely determined by three angles, or *Degrees of Freedom* (3-DOF): **pitch**, **roll** and **yaw**. Figure 1.3 is a visualisation of these three angles as rotations around their corresponding axes. In AR, yaw can be seen as the direction an observer is facing while pitch and roll can be seen as the way an observer is tilted.

Magneto-resistive sensors, or **magnetometers**, are used as an "electronic compass" and are used to determine **yaw** or **direction**. Newer generations of electronic compasses use three axes to ensure a correct reading no matter the tilt of the sensor. Wikitude and Layar rely on this sensor to determine the direction the user is facing in order to correctly overlay information on top of landmarks.

Accelerometers measure *proper acceleration*, which is the acceleration experienced by an object relative to an object in free fall. An accelerometer at rest will register 1 g (approximately 9.81 m/s^2) upwards because a stationary object on the Earth's surface is accelerating upwards relative to an object in free fall. Accelerometers can not determine yaw (direction), but they can be used to determine pitch and roll.

Electronic **gyroscopes** can be used to determine all three degrees of freedom of orientation at once. Some cutting-edge smart-phone devices are now being released with embedded Micro Electro-Mechanical System (MEMS) gyroscopic sensors [23]. A common problem in visual tracking systems is that sudden movements of the camera can cause image blur which may reduce, or inhibit, further visual tracking. A device with both a gyroscope and accelerometer will be able to determine direction accurately, as well as detect and compensate for loss of visual tracking when sudden movements occur [22].

A *hybrid* tracking system utilising computer vision tracking as well as gyroscope and an accelerometer is well suited for robust, real-time AR development [22], because orientation sensors can be used to quickly and accurately determine three of the six degrees of freedom of pose and help the visual tracking system recover when experiencing image blur and distortion due to sudden movements or rotations.

Video Feed

AR super-imposes information on the view the user is seeing. For dynamic systems where information must be blended with objects in the user's view and the user's view of the real world, a capture device such as a digital camera is required. For AR purposes a real-time camera (video camera) is preferable. Most applications use a single camera and are known as *monocular* systems. *Stereoscopic* systems use two cameras to determine depth and some systems may use several cameras to improve depth estimations.

---

[2] `http://www.wikitude.org/`

[3] `http://www.layar.com/`

Touch Input

On hand-held AR systems such as smart-phones or tablet PCs, the screen may double as an input and an output device. A touch-sensitive layer can capture multiple touch inputs which can be used to manipulate AR objects directly on the screen.

1.3.2 Data Feed

An AR system requires a data feed of information with which to augment a user's reality. The data feed may either be a database directly available to the AR system, or it may be information aggregated from sources available via a network connection. Currently there is much development in using the internet as a data feed for AR purposes. Wikitude and Layar mentioned above is examples of applications using data available on the internet as a data feed.

1.3.3 User Feedback Peripherals

Visual

Currently, augmented reality development is mostly focused on augmenting what the user sees. There are four major types of visual displays available for augmented reality development: Video see-through, optical see-through, virtual retinal displays and projector based displays.

Video see-through displays take images generated by a video camera and blend it with computer generated graphics in order to superimpose images on the captured scene. The user cannot view the real world directly and the user's eyes must be covered by a Liquid Crystal Display (LCD) screen, or the user must use an LCD screen as a "window into a world" device like on a tablet PC or mobile phone.

Optical see-through displays allow the user to see the real world and a superimposed image at the same time. An example of an optical see-through display would be standard spectacles with the ability to display computer generated images by projecting an LCD generated image on a partially reflective mirror or transparent Organic Light Emitting Diode (OLED) display while being able to view the real world directly. An example of a see-through display is shown in figure 2.1. Optical see-through displays are more difficult to implement than Video see-through displays and are therefore more expensive. Both of these displays can be implemented in a Head-Mounted Display configuration.

Virtual retinal displays use a laser or sophisticated directional Light Emitting Diode (LED) technology to draw an image directly on to the user's retina. Though this is a highly interesting technology, perfect for AR purposes [26], there are no viable prototypes or commercial units currently available. We are almost certain that we will see this technology evolve within the next ten to twenty years.

Projection based displays make use of a small video projector to project images onto

nearby surfaces. MIT developed a wearable Augmented Reality system named the "6th sense project" [30] that utilises a wearable projector to overlay computer generated information on the real world.

Audio

With a pair of head- or earphones, it is fairly simple to provide the user with audio information. Speech synthesis can also be used to read information to a user, e.g. words from a text recognition system.

1.3.4 Processing Unit

Processing may be done either *online*, meaning on the device itself, or off-line which means processing is offloaded to a different system. For the purpose of this thesis, any generic processing unit will suffice and we will not elaborate any further. In section 5.3 we compare the processing units of four new smart-phone devices.

1.4 Motivation for a Mobile Markerless AR System

1.4.1 Why Mobile

One of the platforms that is potentially well-suited for AR applications are smart-phone devices. Many new smart-phones include all of the hardware components necessary for AR discussed in section 1.3: a screen as output, a camera as visual input, integrated orientation sensors [23], touch input and increasingly powerful processors and graphical 3-D rendering subsystems. Most importantly, AR capable smart-phones are becoming more ubiquitous every day and therefore a mobile AR application has the potential to be widely distributed and used [38].

1.4.2 Why Markerless

The difference between marker-based AR and markerless AR will be fully discussed in section 2.2. Marker-based AR has been implemented on smart-phones very successfully in the past [16, 26, 3], therefore investigating a marker-based approach would not be very exciting or deliver interesting results. *Markerless* AR techniques, on the other hand, are still being refined on mobile platforms due to greatly increased processing power requirements. We propose that smart-phones are reaching a stage where they are powerful enough to handle the intense processing calculations needed for the implementation of a markerless AR system in real-time. More importantly, we propose that a markerless tracking strategy may be combined with orientation sensors integrated in many new smart-phone devices [23] to form a *hybrid* tracking system that utilises dedicated hardware sensors to aid pose estimation.

1.5 Goals and Objectives

The broader objective of this study is to investigate the challenges involved in the implementation of a hybrid markerless AR system on modern smart-phone platforms. In order to do this, the following specific goals were identified:

- Do a theoretical study and review of existing methods and literature related to AR in general, and AR systems on smart-phones in particular.
- Evaluate various computer vision algorithms and techniques within the context of mobile hardware to determine the suitability of each for mobile AR development.
- Investigate the use of integrated smart-phone sensors in pose estimation.
- Implement a proof of concept AR application on a mobile device.

1.6 Contributions

This thesis gives an overview of AR and the deployment of AR applications on mobile hardware. It investigates the limitations of smart-phones as an AR platform, but also what makes them particularly suited for AR development. The study reported in this thesis makes the following contributions:

- A collection of experiments benchmarking specific computer vision algorithms on mobile hardware.
- A method of combining integrated sensors with visual tracking to aid pose calculation.
- A working proof of concept mobile markerless AR application.

1.7 Document Overview

Chapter 2 discusses the history of AR and the differences between marker-based, model-based and markerless systems.

Chapter 3 provides an introduction to pose estimation, specifically the rotation matrix, $\mathbf{R}$, and translation vector, $\vec{t}$. It also provides a means of estimating $\mathbf{R}$ using data from an integrated accelerometer sensor for use with an OpenGL rendering system and finally gives an overview of a hybrid markerless Augmented Reality system.

Chapter 4 provides more information regarding specific computer vision techniques that are evaluated for use in our prototype hybrid tracking system.

Chapter 5 shows our specific implementation of computer vision, sensor fusion and pose estimation on a mobile platform.

Chapter 6 provides the configurations and results of tests performed to evaluate specific computer vision functions and overall system performance.

Chapter 7 compares this thesis to prior work, reviews our achievement of goals and makes recommendations for further work.

The appendices contain raw sample data from tests and data sets used for testing.

Footnotes, for the most part, only refer to website URLs and may be safely ignored.

Chapter 2

Augmented Reality: A Theoretical Background

2.1 Introduction

In the digital age, man has acquired the ability to adapt, or augment, his perception of his environment with additional information in real-time.

Although the concept of Augmented Reality as we understand it today is only about two decades old [3], there is a much research that is closely related and material available from other fields that AR draws from. One of the catalysts behind the fast growth in the field of AR currently is the availability of algorithms, libraries and frameworks that have been adapted from applications in virtual reality, robotics and general computer vision. We will take a brief look at the history of these supporting research areas.

In his book, "Multimedia: From Wagner to Virtual Reality" [29], Packer introduces Dr Ivan Sutherland, father of the head-mounted display (HMD). He developed a prototype in 1966 and completed a working version by 1968. He nicknamed it "The Sword of Damocles" which can be seen in figure 2.1a. It was so heavy that it had to be suspended from the ceiling. Sutherland said, "A display connected to a digital computer gives us a chance to gain familiarity with concepts not realizable in the physical world. It is a looking glass into a mathematical wonderland." What Sutherland imagined was an incredible "virtual reality" in which we could immerse ourselves and explore concepts not tangible in the real world. HMDs may soon be adapted into transparent spectacle-style AR screens as shown in figure 2.1b.

2.2 Computer Vision Tracking for Augmented Reality Systems

In "Visual Tracking for Augmented Reality", Klein [16] writes: "The primary technical hurdle AR must overcome is the need for robust and accurate *registration*. Registration is the

(a) Ivan Sutherland's first Head-Mounted Display. Courtesy of [29].

(b) Near-future spectacle style see-through AR display.

Figure 2.1: *Two examples of AR Head-Mounted displays.*

accurate alignment of virtual and real images without which convincing AR is impossible...".

The majority of AR systems rely on a camera that is set up in such a way as to capture a view that is closely related to the actual view of the user. Video frames coming from the camera are processed using computer vision techniques, discussed in chapter 4, in order to track the movement of the camera's view. Once the camera's movement can be successfully tracked, further computer vision algorithms are used to solve the problem of registration. Klein shows that accurate and robust registration is possible using cheap and readily available video cameras combined with visual tracking techniques that have been developed over the last two decades.

Visual tracking systems for Augmented Reality generally fall into three categories. *Markerless* tracking systems, *model-based* tracking systems and *marker-based* tracking systems. With marker-based systems, specially prepared markers are placed in an environment and are then detected by an AR system in order to aid registration. Model-based systems can only track 3-D objects that have a known shape and can be detected by an AR system with a model of the object. A markerless approach tries to track natural features in a completely unknown environment to estimate pose. Some reports classify model-based trackers as markerless [16], however we feel that this causes unnecessary ambiguity as a 3-D model is essentially just a more complex form of a marker.

2.2.1 Marker-based Tracking

Pose is defined as the combination of camera rotation and translation relative to a system of axes in real-world coordinates. Pose will be discussed in detail in section 3.1. The fiduciary marker is used as a "point of reference" or "landmark" that can easily be detected

by a computer vision based AR system. A fiduciary marker is usually a printout of an easily identifiable, two dimensional, basic geometric shape like a square. With marker-based systems, fiduciary markers are used to facilitate pose estimation and registration. Once it is detected, the system can calculate pose by examining the projective distortion between the camera image of the marker and the known image of the marker. Marker-based systems are suitable for applications where it is possible to prepare the AR environment beforehand, such as games, medical applications, guided tours, assembly, architectural visualisation, business cards, etc.

The use of fiducial markers for computer vision registration probably evolved from the use of "landmark pins" in the medical world to determine the shape of bones and precise orientation of body parts. Taylor [36] talks about "model-reality" registration by placing markers on a patient at the time of surgery to provide real-time tracking and visualisation of anatomical features. Mellor [27] describes a medical "X-ray vision" system for brain surgeons very closely resembling AR: "Enhanced reality visualization is the process of enhancing an image by adding to it information which is not present in the original image". Mellor accomplished this by placing small printed black rings on a real-world object and determining the positions of the rings off-line with a laser range scanner.

There are various other examples of such *passive* fiduciary markers used in a number of diverse applications over the past two decades. Many of these are described in the comprehensive GIS technologies report titled "Augmented Reality Technology" [26]. Fiducials are not limited to two-dimensional visual shapes, however. *Active* fiduciary markers may be implemented using infra-red or visible-light LEDs [16], magnetic markers and ultrasound. Acoustic trackers have been used since as far back as 1993, and infra-red systems have been used since 1991 [26, 16].

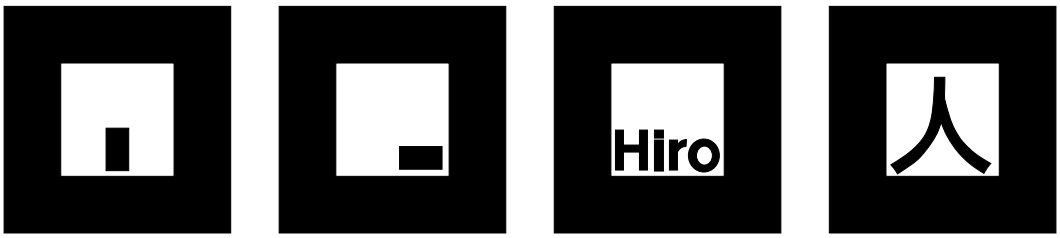
Klein further distinguishes marker-based systems between "inside-out" trackers and "outside-in" trackers:

- "*Inside-out*" trackers are systems as mostly described thus far where fiduciary markers are placed on world objects and the imaging sensor (camera) is attached to the observer to determine the pose of the observer relative to an object. There are essentially one sensor and many fiducials.

- "*Outside-in*" trackers, on the other hand, are systems where one set of fiducials is attached to the observer, and a number of imaging sensors are placed in the environment to track the fiducials connected to the observer. There is essentially many sensors and only one set of fiducials.

One of the most popular libraries that uses passive fiduciary markers and "inside-out" tracking to enable the development of AR applications is ARToolkit[1]. It was developed based on research done by Kato and Billinghurst in their paper "Marker tracking and HMD

[1] `http://www.hitl.washington.edu/artoolkit/`



(a) Sample 1 (b) Sample 2 (c) Hiro (d) Kanji

Figure 2.2: *ARToolkit fiduciary markers. This toolkit has become one of the most widespread marker-based AR systems due to its simplicity and low processing requirements.*

calibration for a video-based augmented reality conferencing system" [15] in which they write: "We propose a method for tracking fiducial markers and a calibration method for optical see-through HMD based on the marker tracking." The library is actively being maintained as an open-source project at SourceForge[2] and commercial licensing is available from ARToolworks[3]. ARToolkit is easy to use, has low processing requirements and is easily ported to multiple platforms including smart-phone devices. Figure 2.2 shows an example of fiduciary markers which are included as samples with the ARToolkit library.

ARToolkit works by finding lines within an image and locating regions that are enclosed by four lines. These regions are extracted and identified using fast template matching. Since it is known that each marker is square, full 3-D pose can be calculated by looking at the distortion of the square's four corners. What sets ARToolkit apart from similar libraries is its ability to process each frame in very little time which allows for high frame-rates. According to ARToolworks, ARToolkit is being used in many projects, with the download count of the library exceeding 160 000 downloads at time of writing.

There have been some improvements on ARToolkit over the past couple of years. Most notably the use of "natural features" in markers as opposed to simple geometric shapes. A "feature" is also known as a "point-of-interest" and is discussed in section 4.3.

Wagner uses computer vision techniques usually associated with markerless systems (such as interest-point description discussed in section 4.4) and applies it to implement a more interesting marker-based system on mobile smart-phone devices. In the ISMAR[4] paper "Pose Tracking from Natural Features on Mobile Phones" [38], Wagner describes an AR system that claims to be the "first fully self-contained **natural feature tracking** system capable of tracking full six degrees of freedom (6-DOF) at real-time frame rates (20 Hz) from natural features using solely the built-in camera of the phone." The term "natural feature tracking" usually implies a markerless approach, however Wagner's system requires the use of textured

[2] `http://artoolkit.sourceforge.net/`

[3] `http://www.artoolworks.com/`

[4] `http://www.ismar-society.org`

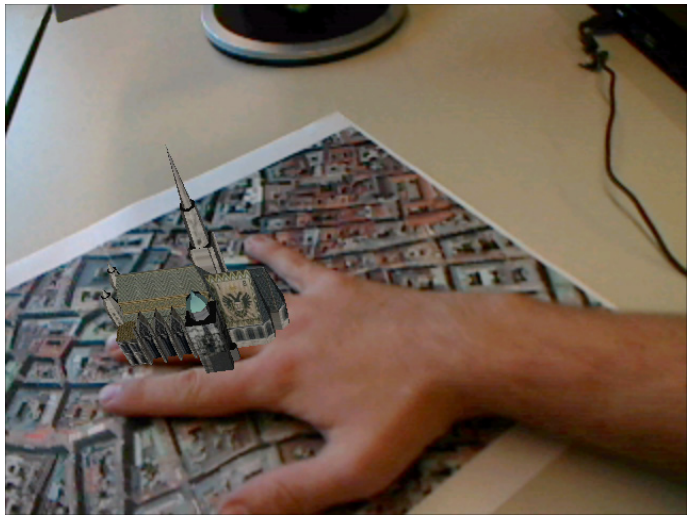


Figure 2.3: *"Natural Feature Tracking" system described by Wagner. Picture courtesy of [38].*

(a) Artwork framed by AR fiduciary marker.

(b) ISO/IEC16022 "Data-Matrix" 2-D barcode inside fiduciary marker.

Figure 2.4: *Studierstube Tracker advanced fiduciary markers.*

planar targets which are known beforehand and which are used to create training data sets. The features that are tracked are therefore inherently *unnatural* and we feel that Wagner uses the word "natural" erroneously as this system is definitely marker-based. Despite this incongruity, the system is very impressive.

An example of one of Wagner's markers is shown in figure 2.3 which demonstrates the system's resilience towards *occlusion* - a major problem for simpler fiduciary markers consisting of basic geometric shapes. Because simple markers usually detect areas bounded by four edges, it fails when one of the edges is broken. A marker based on features can easily recover from occlusions that are introduced as long as there are enough features that are still visible. In figure 2.3, an AR object (church) is superimposed on top of a marker (cityscape) with a hand covering a large part of the marker.

The Christian Doppler Laboratory[5] has recently released an innovative marker-based tracking library named "Studierstube Tracker". This tracker allows tracking of more complicated geometric shape markers such as markers with graphics or even "DataMatrix" data inside the marker. Examples are shown in figure 2.4. We will not be using marker-based tracking in this thesis, as discussed in section 1.4.

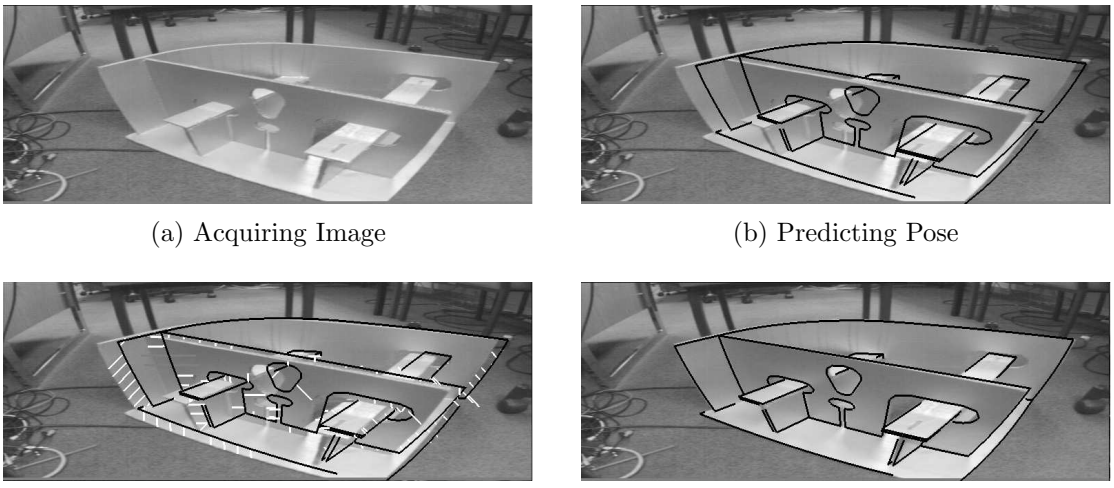
2.2.2 Object-based Tracking

Object-based tracking is similar to marker-based tracking in that it is aided by an object with known shape. Usually the system makes use of a 3-D Computer Aided Design (CAD) model of a known object in order to identify its pose and calculate the depth of points on its surface. A known object is identified either by shape matching or texture matching if the surface texture of the shape is also known. Similar to marker-based systems, object-based trackers can only be used in environments or with objects that have been prepared beforehand. One interesting application is the visualisation of historic architecture where a historic rendition of an area may be overlaid on a current view of the area.

One of the earliest object-based tracking systems is known as RAPiD (Real-time Attitude and Position Determination) and was described by Harris [12]. The processing capacity of computers two decades ago were far less than what is available today. To overcome processing limitations, Harris focused on minimising the amount of information extracted from video frames that had to be processed. RAPiD requires a CAD model of the object to be tracked. Each frame, the pose of the object is predicted using a motion model of the system. The CAD model's edges are projected onto the image for that frame according to the predicted pose. By using an appropriate motion model, updating frequently and given that the object will not move around too much between frames, the predicted pose error can be kept small. Control points along the edges of the model are used to do small localised searches for actual edges in the image perpendicular to the control points. There are usually 20 to 30 such control points per frame. The difference between predicted edges and actual edges are found, and the error is minimised using a least-squares method. By heavily restricting the search for edges and not performing full-frame edge detection RAPiD was able to operate in real-time at 50 Hz.

Klein [20] describes a system that uses a model-based approach very similar to RAPiD, but with the addition of rate gyroscope sensors that enable tracking even if the video feed exhibits considerable motion blur, as may be the case with a head-mounted camera. Their system operates in real-time at 50 Hz and projects graphics onto an optical see-through display. Figure 2.5 is a visualisation of how this system, and other RAPiD style systems function. In figure 2.5a, a grey-level frame is acquired from the video feed. In figure 2.5b, the edges of the CAD model of the object is projected onto the image according to the pose as

[5] `http://studierstube.icg.tu-graz.ac.at`



(a) Acquiring Image (b) Predicting Pose

(c) Finding Errors (d) Minimising Errors

Figure 2.5: *Model-based AR tracker. Pictures courtesy of [20].*

predicted by a motion model. In figure 2.5c, the difference between model edges and actual edges are found. Finally, the pose is corrected and updated in 2.5d.

I was fortunate enough to follow a three month internship at the Katholieke Universiteit Leuven. During this time we were introduced to the "Aerial Crowd" project[6]. This is an ongoing research project and no papers have been published yet. Its objective is to "take Augmented Reality out of the laboratory and into a real urban environment, where it can be used to virtually add or remove buildings and crowds." Essentially, it combines a video feed from an aerial vehicle with a video feed from a ground-level camera to monitor the movement of people on the ground. Then it augments the aerial shot with additional people and buildings in real-time. This may aid city and event planners to visualise the movement of people and simulate the effects of new structures.

The Aerial Crowd system uses a model based approach but relies on textured models as opposed to model edges. Just like the feature-based system in subsection 2.2.1, the tracker attempts to find and match features from a video feed to a known texture. Once a match is found, the depth of the point at that feature can be retrieved from the textured model, as the position of points on the model are known. By finding a number of point depths, the camera pose can be calculated. A textured model is generated using the "Arc3D" web-based 3-D reconstruction service described by Vergauwen [37]. A series of images of an object from different points of view are submitted to the webservice and a Meshlab[7] 3-D model file is returned. Since a textured model based AR system relies on the 3-D reconstruction of an arbitrary object, and a 3-D reconstruction of such an object can be made from a video sequence panning over the object, combining these two steps should enable the creation of a

[6] `http://www.vision.ee.ethz.ch/research/projects_icu.cgi`

[7] `http://meshlab.sourceforge.net/`

fully *markerless* tracking system. Unfortunately the technique used by the Arc3D system is very slow, executing over several hours. Ongoing research in computer vision based tracking attempts to achieve the same 3-D reconstruction techniques combined with feature matching in real-time.

2.2.3 Markerless Tracking

Marker-based and object-based systems are essentially solved problems as discussed in subsection 2.2.1 and subsection 2.2.2. Even though some refinements and enhancements are still being developed, these systems have been implemented with good accuracy and robustness on devices with low processing capacity [16]. While they are well suited for cases where a prepared environment or object is explored, they are not appropriate for use in new and unknown environments. *Markerless* tracking systems attempt to identify and track features that naturally occur in the environment across consecutive video frames. By detecting and matching features across frames the system can use a combination of triangulation, filtering and bundle-adjustments to determine camera pose. This is far more computationally intensive than a marker-based and object-based approach.

According to Klein [16], markerless AR systems are based on techniques developed in the field of robotics where a robot is expected to explore an unknown environment (without *a priori* knowledge). One of the most well-known and widely investigated [9, 39, 17, 14, 40, 8] techniques used to accomplish this is known as *Simultaneous Localisation and Mapping* (SLAM). **Localisation** is the real-time estimation of pose within an environment relative to a map of the environment. **Mapping** is the real-time generation of a map or model of an environment given a set of sensor samples. However, to build a good map representation of the environment, a good estimation of pose is required. Mapping and Localisation can be seen as a "chicken and egg" problem, they are tightly coupled, each step being dependent on the other. In order to solve this inter-dependency, SLAM makes use of various techniques such as filtering based on adaptable models and using an iterative approach to constantly improve previous estimates [20]. The definition of SLAM can be found on the OpenSLAM[8] website for researchers, but it is roughly defined as the problem of building an initial model of the environment to create a new map and then iteratively improving it while simultaneously localising the observer within that map. Some techniques used to reduce errors are using key frames and investigating the seams between parts of the map to ensure that they fit together. Markerless tracking, as well as some of the above-mentioned techniques, are discussed further in chapter 4.

Davison [9] describes a real-time SLAM based system for a single camera moving through unknown scenes. He presents a "top-down Bayesian framework" for *monocular* (single-camera) localisation through the mapping of a sparse set of natural features using structure from motion techniques. Structure from motion refers to the process of finding the three-

[8] `http://www.openslam.org/`

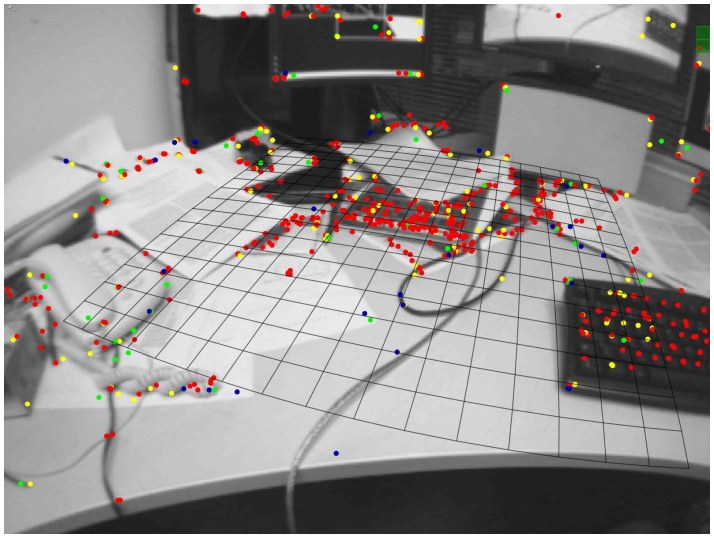


Figure 2.6: *A desktop tracked using PTAM at 55 FPS. Ground plane already estimated after initialisation. Picture courtesy of [18].*

dimensional structure of an object by analysing tracked points over time using a single camera, but structure can also be determined instantaneously using two cameras in a *stereoscopic vision* configuration. Davison's system still requires four known features at start-up in order to initialise the system, but from there on 3-D positions can be calculated from newly detected and tracked points in real-time.

Klein [18] introduces a new approach to SLAM, namely *Parallel Tracking and Mapping* (PTAM). He presents a very successful method for estimating camera pose in a completely unknown scene with no "helper" markers to help initialise the system. PTAM has been designed from the ground up specifically to track a hand-held camera in small AR workspaces. In contrast to SLAM where localisation and mapping is very closely linked, PTAM splits tracking and mapping into two separate tasks that it processes in parallel threads. One task focuses on robustly tracking erratic motion from a hand-held camera while the other task produces a 3-D map of point features tracked from previous video frames. This parallelisation allows for the use of optimisation techniques that can create detailed maps and state-of-the-art robust tracking in real-time. An impressive demonstration application and video is available on Klein's website: `http://www.robots.ox.ac.uk/~gk/PTAM/` . Figure 2.6 shows PTAM tracking a desktop and indicates the estimated ground plane. The system in this figure is tracking 660 interest points at 55 frames per second.

Klein [19] later adapted PTAM to operate on mobile hardware, specifically on an iPhone 3G. This was very successful in terms of speed and accuracy. He manages to obtain robust, full markerless tracking on mobile hardware in real-time. Figure 2.7 shows PTAM running on an iPhone, first initialising and estimating a ground plane, then adding map points and finally drawing AR graphics on top of the ground plane. This work by Klein is far ahead of any other research we have been able to find on both mobile and computer platforms.

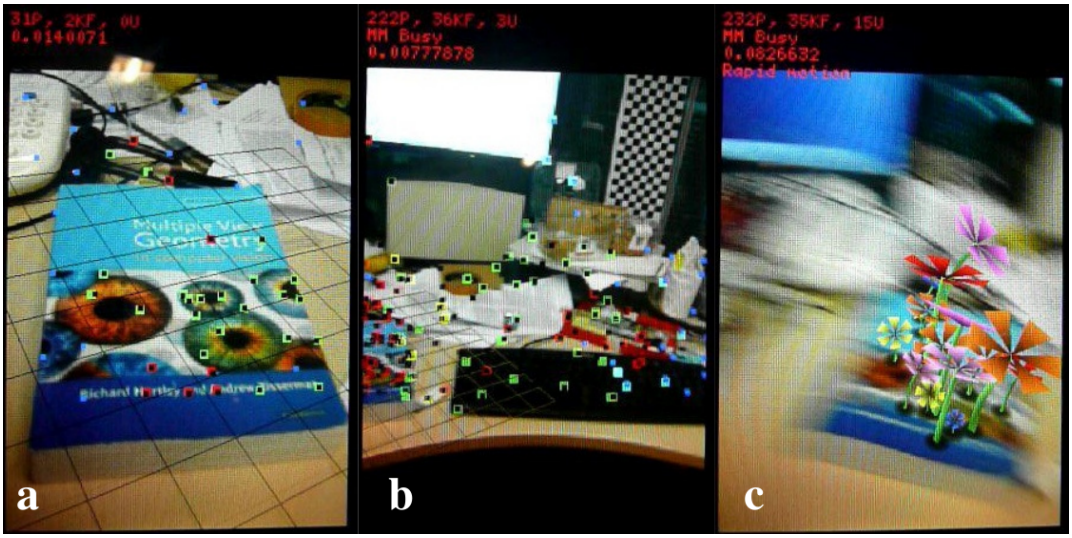


Figure 2.7: *PTAM running on an iPhone 3G. First a map is initialised, next the map is expanded to about 220 points, finally AR objects are inserted. Picture courtesy of [19].*

The feat of implementing it on an iPhone 3G is made even more impressive by the iPhone 3G never officially supporting video frame capture. In order to achieve real-time framerates, Klein has put a lot of effort into managing the cost of bundle adjustments, requiring fewer map points, requiring fewer key-frames and simplifying stereo initialisation.

Klein's mobile PTAM uses purely visual tracking to determine camera pose. In his paper on mobile PTAM he mentions that this work opens up a number of interesting avenues for further research, specifically exploiting hardware orientation sensors such as accelerometers, magnetometers and GPS sensors present in most new smart-phones. In his conclusion, Klein writes: "These extra capabilities could surely be used to benefit visual tracking and AR." This thesis will focus on exploiting these extra capabilities for AR on mobile devices.

2.3 Summary

In this chapter we provided a brief overview of the history of AR development and the three types of visual tracking systems. In the following chapters we will discuss the theory behind visual pose estimation, accelerometer aided orientation estimation, specific computer vision techniques and our own implementation of a proof of concept AR system.

Chapter 3

System Overview

3.1 Introduction

When designing an Augmented Reality system, the most important goal is keeping track of the system's pose within a world-coordinate system in real-time. As already discussed in chapter 2, pose can be determined using only computer vision, only integrated sensors, or the combination of vision and sensors in a *hybrid* configuration. Pose consists of six degrees of freedom (6-DOF) in three dimensional space. Three of the six degrees of freedom are contained in a rotation matrix, $\mathbf{R}$, and the other three in a translation vector, $\vec{t}$. Figure 3.1 shows a red cube centred on the world-coordinate system and a camera that has been rotated and translated relative to the origin of the world-coordinate system, facing the cube. The axis of the camera pointing at the cube is the camera's z-axis. As discussed in subsection 5.5.4, translation relative to the camera's z-axis is equivalent to a change in scale. The relationship between the camera's local coordinate system and the world-coordinate system is given by $\mathbf{R}$ and $\vec{t}$.

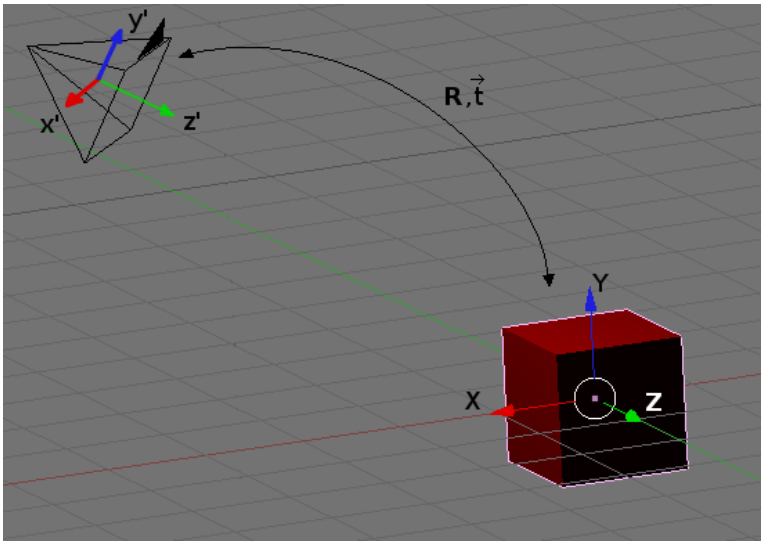


Figure 3.1: *Visual representation of* $\boldsymbol{R}$ *and* $\vec{t}$.

Before we discuss the specific computer vision techniques used in the implementation of our prototype application for finding $\mathbf{R}$ and $\vec{t}$, we must provide a brief overview of the fundamentals of computer vision, specifically the camera model, the homography transform and the fundamental matrix.

3.2 Pinhole Camera Model

"Vision" is the perception of light reflected off of objects. For the purpose of visualising this effect, we may imagine that light is made up of a bundle of light-rays. Figure 3.2 shows a single ray of light, emanating from a light-source (the sun), reflecting off an object (a tree) and into an eye. The eye may be replaced with a camera as they perform exactly the same function: sampling incoming light-rays to form a two-dimensional representation of the world. This representation is known as an *image* and the two-dimensional array of sensors doing the sampling is known as an *imager*, according to Bradski [6].

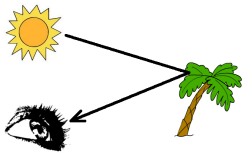


Figure 3.2: *Visualising a single ray of light.*

In "Learning OpenCV", Bradski [6] suggests investigating a simple pinhole camera model in order to understand how an image is formed. Figure 3.3 shows a representation of a simple pinhole camera model. In the figure, light reflecting off an object on the right with height H passes through a small hole in the centre of a "pinhole plane" and is *projected* onto an imager on the left. The object is at a distance of Z from the plane and the imager is at a distance of f from the plane. Due to the geometry of this configuration, the object appears upside down on the imager with a reduced height, h. With the rule of similar triangles, h

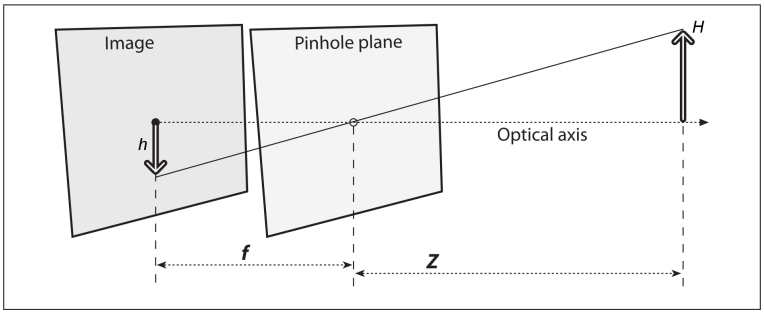


Figure 3.3: *A pinhole camera model courtesy of [6].*

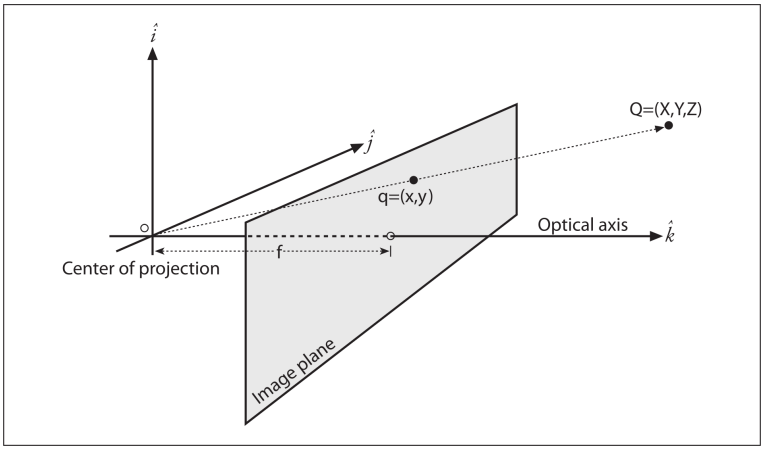


Figure 3.4: *Definition of the image plane courtesy of [6].*

---

can be found with:

$$h = -f\frac{H}{Z} \tag{3.1}$$

The size of h relative to H is determined by the distance of the object from the plane as well as the *focal length* of the camera defined as f. The small hole (aperture) is known as the *centre of projection* and the line perpendicular to the pinhole plane running through the aperture is known as the *optical axis* as shown in figure 3.3.

To simplify the mathematics, we rearrange the configuration by moving the imager *in front* of the centre of projection so that the image is no longer upside down, as shown in figure 3.4. This is not physically realisable, so we define a figurative *image plane* as a plane in front of the centre of projection at a distance of f. A point on a real object, $Q = (X, Y, Z)$, is projected onto a point, $q = (x, y)$, on the two-dimensional image plane at f by the ray that is reflected off Q, passes through the image plane and terminates at the centre of projection. This is equivalent to the model in figure 3.3 except that the projected image is no longer upside down. The height of the projected object, h', in the image relative to the real height of the object is now given by:

$$h' = f\frac{H}{Z} \tag{3.2}$$

Due to imperfections in the manufacturing process of making a camera, the imaging sensor is never perfectly centred on the aperture (centre of projection). We introduce two parameters, c_x and c_y, to model the offset of the imaging sensor relative to the aperture in the x- and y-directions. In addition to this imperfection, the shape of pixels on an imager is not perfectly square but rather rectangular. Even though the focal length is the same in the x- and y-directions, this imperfection may be modelled by assigning a different variable for the focal length in the x-direction, f_x and in the y-direction, f_y. In order to find the x- and y-coordinates on an image of a point Q in physical space projected onto an imperfect

imager, we can use the equations:

$$x_{image} = f_x \frac{Q_X}{Z} + c_x, \quad y_{image} = f_y \frac{Q_Y}{Z} + c_y \tag{3.3}$$

Where x_{image} and y_{image} are pixel positions on an image.

This relationship between a point Q in the physical world and a point q on the image is known as a *projective transform*. For simplicity we will use *homogeneous coordinates* to represent q. Since q is a two-dimensional vector and Q a three-dimensional vector, we add one other non-zero "dimension", $\tilde{w}$, to q to form $\tilde{q}$ so that $\tilde{q}$ and Q have the same number of dimensions as shown in equation (3.4). The only impact this has on q is that we need to divide $\tilde{x}$ and $\tilde{y}$ by $\tilde{w}$ to find their true values after calculating $\tilde{q}$. We define a 3-by-3 *camera intrinsics matrix*, $\mathbf{M}$, that contains the parameters representing camera imperfections as shown below. Q, $\tilde{q}$ and $\mathbf{M}$ are defined as follows:

$$\tilde{q} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}, \quad \mathbf{M} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.4}$$

With $\tilde{q}$ and Q represented in homogeneous coordinates, $\tilde{q}$ can now be calculated as:

$$\tilde{q} = \mathbf{M}Q \tag{3.5}$$

So that:

$$\mathbf{M}Q = \begin{bmatrix} f_x X + c_x Z \\ fyY + cyZ \\ Z \end{bmatrix} = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ fy\frac{Y}{Z} + cy \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix} = \tilde{q} \tag{3.6}$$

Once $\tilde{q}$ is found, we find that $\tilde{w} = Z$ and we can find the true pixel positions:

$$x_{image} = \frac{\tilde{x}}{\tilde{w}}, \quad y_{image} = \frac{\tilde{y}}{\tilde{w}} \tag{3.7}$$

In practice, the positions of pixels in an image must be corrected before they can be used for 3-D reconstruction because they are influenced by lens distortions caused by inherent imperfections in the camera and lens manufacturing processes. We will not cover these distortions in details and will assume a perfect camera-lens assembly. The reader may explore lens distortion in more detail in "Learning OpenCV" [6].

3.3 Planar Homographies

In Computer Vision, two images taken of a two-dimensional surface from different angles are related by a *homography*. Points on one image can be mapped to points on the other by a homography transform. Since the camera imager is a two-dimensional surface, points on a two-dimensional surface in the real-world can be mapped to pixels in an image using

a homography transform. This has many practical applications in rectification, registration and estimation of camera pose. In section 3.1 the rotation matrix, $\mathbf{R}$, and translation vector, $\vec{t}$, are introduced along with a visual representation of a camera that is rotated and translated in six degrees-of-freedom relative to a world coordinate system. The rotation matrix and translation vector can also be used to determine the rotation and translation of one plane relative to another.

The rotation matrix is a 3-by-3 matrix and the definition is somewhat complex, having various representations. We recommend the reader study "Multiple View Geometry" by Hartley and Zisserman [13] for a more complete explanation. The translation vector, $\vec{t}$, is simply the difference between the positions of two points in 3-D space:

$$\vec{t} = Q_1 - Q_2 = \begin{bmatrix} X_1 - X_2 \\ Y_1 - Y_2 \\ Z_1 - Z_2 \end{bmatrix} = \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix}$$

According to Bradski [6], for a point in the real-world, Q, and a point on the image, q, we can rewrite them in homogeneous coordinates as:

$$\tilde{Q} = \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{Z} \\ 1 \end{bmatrix}, \quad \tilde{q}' = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{bmatrix} \tag{3.8}$$

Note that $\tilde{w}$ from equation (3.4) is set to 1. $\tilde{w}$ only influences scale, and in the following equations we will use a scale factor in our equations to take scaling into account. We combine $\mathbf{R}$ and $\vec{t}$ within a single 3-by-4 matrix:

$$\mathbf{W} = \begin{bmatrix} \mathbf{R} & \vec{t} \end{bmatrix} = \begin{bmatrix} \vec{r_1} & \vec{r_2} & \vec{r_3} & \vec{t} \end{bmatrix} \tag{3.9}$$

where $\vec{r_1}$, $\vec{r_2}$ and $\vec{r_3}$ are the three column vectors of $\mathbf{R}$. The relationship between $\tilde{q}'$ and $\tilde{Q}$ is:

$$\tilde{q}' = s\mathbf{M}\mathbf{W}\tilde{Q} \tag{3.10}$$

where s is a scale factor and $\mathbf{M}$ the camera intrinsics matrix defined in equation (3.4).

Since we are working with planar homographies, $\tilde{Q}$ is not just any point in 3-D space but is specifically a point on a plane in 3-D space. If we choose this plane as $Z = 0$ and $\tilde{Q}'$ as a point on the plane, then $\tilde{Q}'$ can be rewritten as:

$$\tilde{Q}' = \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{3.11}$$

Using equation (3.11) we can simplify (3.10) as follows:

$$\tilde{q}' = s\mathbf{M}\mathbf{W}\tilde{Q}' = s\mathbf{M}\begin{bmatrix}\vec{r_1} & \vec{r_2} & \vec{r_3} & \vec{t}\end{bmatrix}\begin{bmatrix}X\\Y\\0\\1\end{bmatrix} = s\mathbf{M}\begin{bmatrix}\vec{r_1} & \vec{r_2} & \vec{t}\end{bmatrix}\begin{bmatrix}X\\Y\\1\end{bmatrix} \tag{3.12}$$

Even though the third column of $\mathbf{R}$ is no longer necessary, the necessary rotation information is still contained in the other two columns. The 3-by-3 homography matrix, $\mathbf{H}$, is then defined as:

$$\mathbf{H} = s\mathbf{M}\begin{bmatrix}\vec{r_1} & \vec{r_2} & \vec{t}\end{bmatrix} \tag{3.13}$$

where:

$$\tilde{q}' = s\mathbf{H}\tilde{Q}'' \implies \begin{bmatrix}\tilde{x}\\\tilde{y}\\1\end{bmatrix} = s\mathbf{H}\begin{bmatrix}X\\Y\\1\end{bmatrix} \tag{3.14}$$

with:

$$\tilde{Q}'' = \begin{bmatrix}X\\Y\\1\end{bmatrix} \tag{3.15}$$

.

The homography matrix contains the rotation and translation information that relates the positions and orientations of two planes to each other, as well as the projective transform information ($\mathbf{M}$) that relates points in real space to points on an image. $\mathbf{R}$ and $\vec{t}$ are known as *extrinsic* parameters while the parameters in $\mathbf{M}$ are known as *intrinsic* parameters. Bradski explains how $\mathbf{H}$ for a specific camera can be found using multiple images of a special, known planar marker.

3.4 Epipolar Geometry and the Fundamental Matrix

We follow the approach outlined by Bradski [6] in his description of epipolar geometry. If we have two images of a point in the world taken from two slightly different angles, we can use *stereo vision* to find the 3-D position of the point. Stereo vision does not necessarily require two cameras (imagers), a single camera can be used to take both images if the camera is moved between taking the first and second image. Systems using one camera to obtain stereo pairs are known as *monocular* systems in Computer Vision. The geometry relating such a stereo image pair to each other is known as *epipolar geometry*. Figure 3.5 shows such a configuration where a point in space, P, is projected onto two different image planes, $\prod_l$ and $\prod_r$ as the projected points p_l and p_r. O_l and O_r are the centres of projection for each of the imagers that produced the images. e_l and e_r are known as the *epipoles*. e_l is defined

as the projection of O_r onto $\prod_l$ (the image of O_r on $\prod_l$). The lines $p_l e_l$ and $p_r e_r$ are called *epipolar lines*. The *epipolar plane* is defined as the plane on which P, O_l and O_r lie.

If we had only one image, $\prod_r$, we would know that P lies somewhere on the line PO_r in 3-D space. Since we have another image of P on $\prod_l$, we can see from figure 3.5 that the epipolar line $p_l e_l$ is the projection of PO_r onto $\prod_l$. We can find the position of P in 3-D space by using the information given by O_l, $p_l e_l$ and p_l and triangulation.

The difficult part is finding the epipoles, e_l and e_r. They can only be found if we know the relative translation and rotation between the two cameras that produced $\prod_l$ and $\prod_r$. The **essential matrix**, **E** defined in (3.25), is used to relate corresponding points in two images in *world coordinates*. The **fundamental matrix**, **F** defined in (3.30), combines the essential matrix and intrinsic camera parameters to relate corresponding points in *image coordinates* (pixel space). Both are closely linked to the homography matrix, **H**, discussed earlier. To find the essential matrix:

Define the origin of the world-coordinate system to be at O_l. $\vec{P_l}$ is the location of P relative to O_l and the location of the other camera relative to O_l is given by a translation vector, $\vec{T}$. The location of P relative to the origin of the other camera at O_r is given by $\vec{P_r}$. The relationship between $\vec{P_l}$ and $\vec{P_r}$ is then:

$$\vec{P_r} = \mathbf{R}(\vec{P_l} - \vec{T}) \tag{3.16}$$

or

$$(\vec{P_l} - \vec{T}) = \mathbf{R}^{-1}\vec{P_r} \tag{3.17}$$

where **R** is the rotation matrix. The most important step is finding the epipolar plane on which P, O_l and O_r lie. In 3-D space, a plane can be defined by specifying a point on the plane and a normal vector to the plane. A plane can be expressed by

$$\vec{n} \cdot (P - P_0) = 0 \tag{3.18}$$

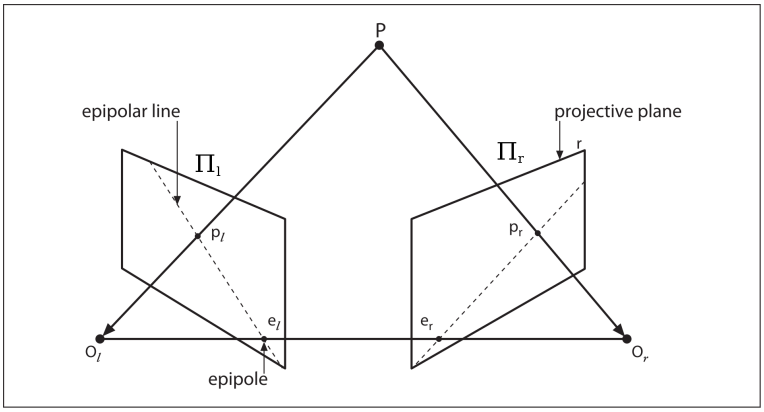


Figure 3.5: *Illustration of epipolar geometry configuration. Picture courtesy of [6].*

where $\vec{n}$ is a normal vector on the plane, P_0 is a known point on the plane and P is any other arbitrary point on the plane. Figure 3.5 shows that $\vec{P_l}$ and $\vec{T}$ (the centre of projection O_r) both physically lie on the epipolar plane and we can use $\vec{T} \times \vec{P_l}$ as a vector normal to the plane. For the epipolar plane we can write equation (3.18) as:

$$(\vec{T} \times \vec{P_l}) \cdot (\vec{P_l} - \vec{T}) = 0 \tag{3.19}$$

or in matrix multiplication form:

$$(\vec{P_l} - \vec{T})^T (\vec{T} \times \vec{P_l}) = 0 \tag{3.20}$$

Substituting (3.17) and using $\mathbf{R}^{-1} = \mathbf{R}^T$ since $\mathbf{R}$ is an orthogonal matrix, we have:

$$(\mathbf{R}^T \vec{P_r})^T (\vec{T} \times \vec{P_l}) = 0 \tag{3.21}$$

Rewriting the cross product in matrix multiplication form by defining a new matrix $\mathbf{S}$ as the matrix representation of a cross product with $\vec{T}$ so that:

$$\vec{T} \times \vec{P_l} = \mathbf{S}\vec{P_l} \tag{3.22}$$

and

$$\mathbf{S} = \left[\vec{T}\right]_\times = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix} \tag{3.23}$$

Substituting (3.22) into (3.21) yields:

$$(\vec{P_r})^T \mathbf{R}\mathbf{S}\vec{P_l} = 0 \tag{3.24}$$

The **essential matrix**, $\mathbf{E}$ is defined as the product $\mathbf{RS}$ so that we have:

$$(\vec{P_r})^T \mathbf{E}\vec{P_l} = 0 \tag{3.25}$$

We see that this equation relates the observed position of P relative to the camera at O_l to the observed position of P relative to the camera at O_r in real-world space. In order to translate this result to pixel space we need to take into account the intrinsic parameters of both cameras, $\mathbf{M}_l$ and $\mathbf{M}_r$, as defined in equation (3.4). From equation (3.5) we have:

$$p = \mathbf{M}P \tag{3.26}$$

Where P is a point in space and p is its projection on an image in pixel space. Substituting $\vec{p_r}$, $\vec{P_r}$ and $\mathbf{M}_r$ into the above equation:

$$\vec{p_r} = \mathbf{M}_r \vec{P_r} \tag{3.27}$$

we can write:

$$\vec{P_r} = \mathbf{M}_r^{-1} \vec{p_r} \tag{3.28}$$

Substituting (3.27) and (3.28) into (3.25) we have:

$$(\vec{p_r}\mathbf{M}_r^{-1})^T\mathbf{E}\mathbf{M}_l^{-1}\vec{p_l} = \vec{p_r}^T(\mathbf{M}_r^{-1})^T\mathbf{E}\mathbf{M}_l^{-1}\vec{p_l} = 0 \tag{3.29}$$

We define the **fundamental matrix**, **F**:

$$\mathbf{F} = (\mathbf{M}_r^{-1})^T\mathbf{E}\mathbf{M}_l^{-1} \tag{3.30}$$

so that (3.29) becomes:

$$\vec{p_r^T}\mathbf{F}\vec{p_l} = 0 \tag{3.31}$$

This constraint gives us a means of calculating the relationship between two images to each other in the form of **R** and **S**. The components of the translation vector, $\vec{T}$, is contained in **S** and is equivalent to $\vec{t}$. From now on we will use $\vec{t}$ to denote the translation vector. In the following section we will discuss how the fundamental matrix can be used to estimate the pose of the two imagers that provided the images relative to each other.

3.5 Computer Vision based Pose Estimation

The fundamental matrix consists of the parameters that describe the relationship between two image planes as discussed in section 3.3. If we can calculate the fundamental matrix, we can determine the relative change in *pose* between two images. In a monocular system, this will allow us to *track* the pose of the camera between frames. Unfortunately we can not determine pose using a single point. In reality we need to match the positions of at least eight points [6] in two images to be able to solve **F** as a linear system of equations. Describing and solving this system is discussed in far more detail in "Learning OpenCV" [13]. Matching points between images is discussed in more detail in chapter 4.

One drawback of using the fundamental matrix to estimate pose is that the process of calculating **F** is highly sensitive to noisy samples [6] . A common means of reducing the effect of noise is to use *outlier detection* to determine whether a reading lies outside an envelope of acceptable values. Common methods of outlier detection are RANdom SAmple Consensus (RANSAC) [10] and Least Median of Squares (LMedS) [33] line fitting. These methods fall outside the scope of this thesis and are not used in our implementation of a prototype AR application.

We do not use computer vision in this thesis to estimate full pose. Instead we will only use computer vision to estimate $\vec{t}$ and rely on integrated sensor readings to find **R**. Our method for estimating $\vec{t}$ using computer vision tracking is described in section 5.5.4. A method for finding **R** by using data from an accelerometer sensor is discussed in the following section.

3.6 Orientation Estimation from Integrated Sensors

In the above couple of sections a simplified overview of pure computer vision tracking is provided. The two most important variables we are interested in for pose tracking is the

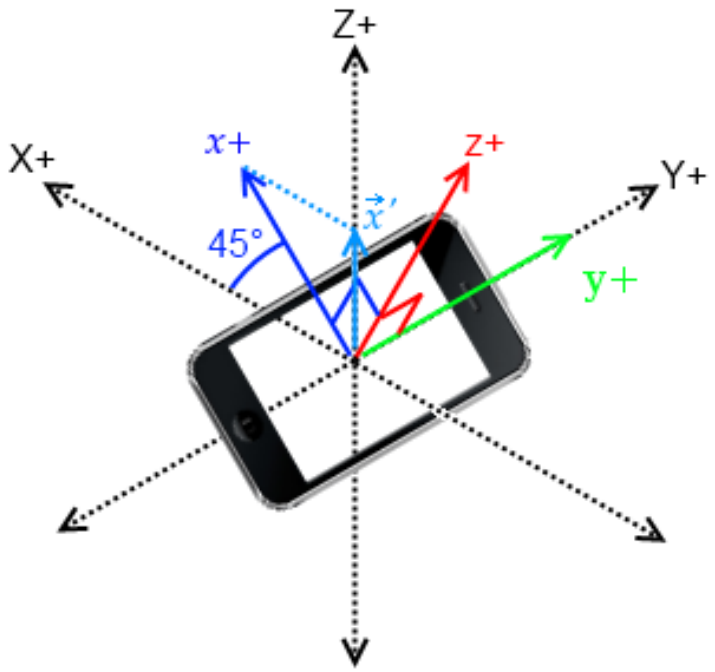


Figure 3.6: *Visualisation of a 45 degree rotation around a single axis of an iPhone.*

rotation matrix, **R**, and translation vector, $\vec{t}$. In this thesis we propose that a hybrid tracking system may reduce the computational requirements of a visual tracking system by using integrated sensors to estimate **R** and limit visual tracking to the estimation of $\vec{t}$.

As discussed in section 1.3.1, there exists a number of hardware sensors that may partially or fully describe the three degrees of freedom of orientation. Some of these sensors are being integrated in modern smart-phone devices [23] and may be beneficial in advancing Augmented Reality on smart-phones as a platform. By the time we started work on this thesis, many smart-phone models already featured integrated accelerometer sensors that are capable of determining two of the three angles of **R**. We will investigate using an accelerometer to determine orientation in this section. Towards the end of our study, the iPhone 4 was released and is one of the first mobile phones to include a MEMS gyroscope in addition to an accelerometer. Lane [23] presents a survey of integrated sensors in smart-phones and discusses the new, growing class of sensor-enabled phones including the iPhone 4. We leave the use of integrated gyroscopes to determine all three degrees of freedom of orientation for future work as discussed in section 7.4.

3.6.1 Accelerometer Aided Orientation

Working with Accelerometer Values

An accelerometer measures the acceleration of an object relative to an object in free-fall. Therefore, an accelerometer held stationary relative to the surface of the earth will return an

acceleration of $1g$ *upwards* perpendicular to the surface of the earth. We define the Z-axis of the world-coordinate system as upwards, perpendicular to the earth's surface and the X- and Y-axes perpendicular to the Z-axis and to each other. A 3-axis accelerometer with axes z', x', and y', rotated arbitrarily around its three axes will return the projection of each of its axes onto the world Z-axis. Figure 3.6 is a visualisation of a device rotated 45 degrees clockwise around its y-axis. $\vec{x}'$ is the vector returned by the x-axis of the accelerometer. The value of $\vec{x}'$ is the projection of the device's rotated x-axis onto the world Z-axis. The accelerometer that we use in this thesis has unit vectors in the x, y, and z axes of $1g$. To find $\vec{x}'$ for a rotation of 45 degrees as shown in the figure, we use the following trigonometric relationship:

$$\vec{x}' = 1 \sin \theta$$

For $\theta = 45$ degrees, $\vec{x}' = 0.7071$. When the physical accelerometer device is rotated approximately 45 degrees clockwise around its y-axis it returns a value very close to 0.7071 for its x-axis reading. More complex rotations are represented as a combination of the device's rotated axes readings. This combination forms the acceleration vector, $\vec{a}$:

$$\vec{a} = \begin{bmatrix} \vec{x}' \\ \vec{y}' \\ \vec{z}' \end{bmatrix} \tag{3.32}$$

Where $\vec{x}'$, $\vec{y}'$ and $\vec{z}'$ are the readings from the accelerometer's x-, y- and z-axes respectively.

An accelerometer can only determine two of the three degrees of freedom that make up orientation. A rotation only around the gravity vector will not result in a change in the projection of the device's axes onto the world Z-axis and thus result in no change in accelerometer reading. We can only use the accelerometer to calculate two of the three rotation components of $\mathbf{R}$. We will "guess" the third rotation component: direction. As it turns out, this will only influence the direction drawn 3-D models face when $\mathbf{R}$ is used to orientate the OpenGL renderer view. We can manipulate the direction by other means, such as user interaction.

Before we can start constructing $\mathbf{R}$, we need to keep in mind that an accelerometer inside a moving object returns acceleration vectors instead of orientation vectors. We assume that a user of a mobile augmented reality system will move the device around in a gradual manner(section 5.5.3), avoiding sudden changes in direction and sudden accelerations. We are only interested in the *static*, or *stable-state* orientation of the device, not the *transient* changes in acceleration. Finding the static orientation of a moving device in a certain orientation is equivalent to holding the device stationary in that orientation and taking a single reading. We define this steady-state response based on readings from the accelerometer as $\bar{a} = \begin{bmatrix} \bar{x}' & \bar{y}' & \bar{z}' \end{bmatrix}^T$. We can find $\bar{a}$ from multiple readings of $\vec{a}$ by using a simple averaging filter that attenuates transient changes in $\vec{a}$ and stabilises around the stationary reading, $\bar{a}$. The low-pass filter is simply implemented as:

$$\bar{a}_i = 0.1 \times \vec{a}_i + 0.9 \times \bar{a}_{i-1}$$

Where $\bar{a}_i$ is the ith stable state estimation, $\vec{a}$ is the ith accelerometer reading and $\bar{a}_{i-1}$ is the previous stable state estimation. From the equation we can see that the effect of instantaneous accelerometer readings and potential sudden changes in acceleration is attenuated.

Calculation of the Rotation Matrix

When constructing the rotation matrix, $\mathbf{R}$, we need to keep in mind that it will be passed to the 3-D rendering system of the mobile device in order to orientate the view on the screen. The camera captures a **scene** of the real world and we want to superimpose models on the scene so that they appear to be a natural part of the real world. The view of the scene must be rotated so that models are drawn on the screen with the correct orientations. The 3-D rendering system on most phones is OpenGLES based and the current view is determined by a number of matrixes. One of these is the 4-by-4 Model View matrix, $\mathbf{M}_{modelview}$ which determines the view's translation, rotation and scale. We need to construct $\mathbf{R}$ so that when we multiply it with $\mathbf{M}_{modelview}$, the orientation of the OpenGL world will reflect the orientation of the real world. It would be difficult to construct $\mathbf{R}$ without in-depth knowledge of the OpenGL rendering system. Luckily, Apple's developer portal[1] provides a recommendation for constructing $\mathbf{R}$ to achieve this goal. For multiplication with the 4-by-4 Model View matrix, $\mathbf{M}_{modelview}$, the $\mathbf{R}$ matrix must be represented in homogeneous coordinates so that we have:

$$\tilde{\mathbf{R}} = \begin{bmatrix} \vec{r}_1 & \vec{r}_2 & \vec{r}_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.33}$$

where $\vec{r}_1$, $\vec{r}_2$, $\vec{r}_3$ are the three 3-by-1 column vectors of $\mathbf{R}$.

Apple recommends that $\vec{r}_1$ should represent the stable-state gravity vector ($\bar{a}$) as a unit vector:

$$\vec{r}_1 = \hat{a} = \frac{\bar{a}}{||\bar{a}||} \tag{3.34}$$

where $||\bar{a}||$ is the length of $\bar{a}$. Note that $\hat{a}$ does not represent the instantaneous acceleration vector, $\vec{a}$.

$\vec{r}_2$ should be an arbitrary unit vector on the plane perpendicular to $\bar{a}$ defined by:

$$\bar{a}_x * x + \bar{a}_y * y + \bar{a}_z * z = 0 \tag{3.35}$$

Apple recommends arbitrarily choosing $x = 0$ and $y = 1$ so that $z = -\frac{\bar{a}_y}{\bar{a}_z}$. We define this new vector, $\vec{b}$ as:

$$\vec{b} = \begin{bmatrix} 0 & 1 & -\frac{\bar{a}_y}{\bar{a}_z} \end{bmatrix}^T \tag{3.36}$$

[1] http://developer.apple.com/library/ios/samplecode/GLGravity/Introduction/Intro.html

$\vec{r}_2$ is then:

$$\vec{r}_2 = \hat{b} = \frac{\vec{b}}{||\vec{b}||} \tag{3.37}$$

where $||\vec{b}||$ is the length of $\vec{b}$.

Apple then recommends that $\vec{r}_3$ be the cross product of $\vec{r}_1$ and $\vec{r}_2$:

$$\vec{r}_3 = \vec{r}_1 \times \vec{r}_2 = \hat{a} \times \hat{b} \tag{3.38}$$

Substituting (3.34), (3.37) and (3.38) into (3.33) gives:

$$\tilde{\mathbf{R}} = \begin{bmatrix} \hat{a} & \hat{b} & \hat{a} \times \hat{b} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.39}$$

Multiplying $\tilde{\mathbf{R}}$ as recommended by Apple with the model view matrix, $\mathbf{M}_{modelview}$ correctly rotates the OpenGL view to reflect the orientation of the real world in the scene. We can multiply these two matrixes by first initialising the Model View matrix with `glMatrixMode(GL_MODELVIEW); glLoadIdentity();` and then multiplying it with $\tilde{\mathbf{R}}$ (represented only as R in code):
`glMultMatrixf(R);`

3.7 Hybrid Tracking System Overview

The hybrid tracking system that we propose decouples the calculation of $\mathbf{R}$ and $\vec{t}$. We estimate $\mathbf{R}$ using the method described in section 3.6.1. This still leaves $\vec{t}$ to be estimated using computer vision techniques. In order to estimate $\vec{t}$ we need to track the movement of points between frames. Computer vision system generally track points by taking incoming video frames, detecting points of interest within the frames and matching the points of interest between consecutive frames. In our system, we take two incoming video frames, the previous frame and the current frame. In the previous frame we detect points of interest using a corner detector. These points are then matched to points in the current frame using sparse optical flow tracking. An array of vectors are returned representing the movement of points across frames. From this array we estimate $\vec{t}$ using our own approach discussed in section 5.5.4 in the system implementation chapter. In order to implement a hybrid, markerless proof of concept AR application, the following problems must be solved:

- A suitable interest point detector must be found and used.
- A suitable interest point tracking strategy must be found and implemented.
- The translation vector, $\vec{t}$, must be found from tracked point information.
- The rotation matrix, $\mathbf{R}$, must be found from accelerometer data.

- $\vec{t}$ and $\mathbf{R}$ must be applied to the 3-D rendering subsystem to reflect the camera's pose and superimpose Augmented Reality objects on a view of the real world.
- 3-D objects must be rendered.

The specific computer vision techniques used to detect and track corners are discussed in detail in chapter 4 and our own implementation of these techniques are discussed in 5.

3.8 Summary

In this chapter we provided a very brief overview of the theory behind the estimation of camera pose which is vital for correct registration. We also provided a method of finding $\mathbf{R}$ from accelerometer readings. In the following chapters we will discuss the specific computer vision techniques used to find and match points between images and our own method of finding $\vec{t}$ using a hybrid system of partial computer vision tracking combined with integrated sensor data.

Chapter 4

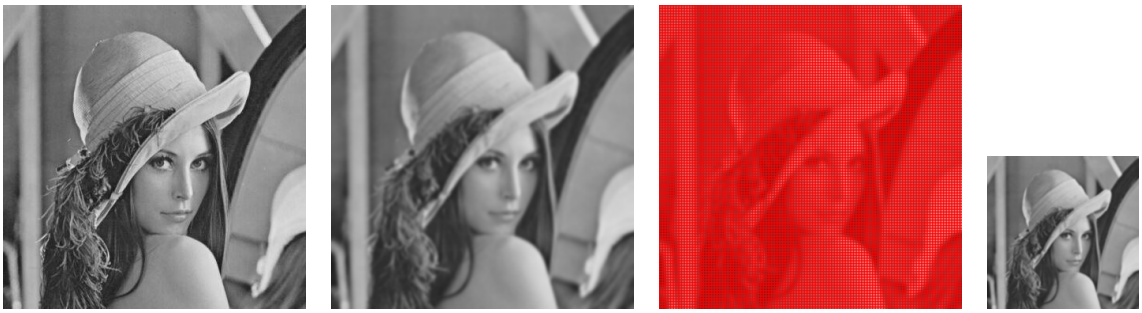
Computer Vision Techniques

4.1 Introduction

In this chapter we will focus on those techniques that may be used for AR tracking. The order in which they are introduced reflects the order in which they may be applied. They are all introduced within the frame of reference of a computer vision tracker system. Image acquisition is the first step in computer vision tracking. Image acquisition is dependent on the system platform. A good assumption to make is that images are acquired as 24-bit, three channel bitmaps, or in other words as an image represented by a combination of its three 8-bit colour components: red, green and blue. We will also assume that images are acquired at 30 frames per second, or 33.3 milliseconds per frame. This assumption is based on the typical form of data that is returned by most cheap, readily available webcams. All of the techniques discussed in this chapter have been tested and timed on our chosen mobile platform, the results are shown in chapter 6.

4.2 Processing of Input Feed

Input feed processing is the second step in computer vision tracking after image acquisition and serves to reduce the amount of data the system needs to handle. A typical camera frame obtained from an inexpensive webcam or integrated mobile phone camera may have a width of 640 pixels and height of 480 pixels. Assuming that a colour image with three channels (red, green and blue) is returned with eight bits per channel, the amount of data per frame equates to $640 \times 480 \times 3 \times 8 = 7\ 372\ 800$ bits of information. Most inexpensive cameras return between 20 and 30 frames per second which means that each frame and all those bits must be completely processed every 33 milliseconds before the next frame arrives. It would be very costly (and unnecessary) to track every pixel retrieved from incoming camera frames and so a number of techniques are used to reduce the amount of processing required in the subsequent stages of tracking. The sample rates of an iPhone 3Gs were tested and the results tabulated in section 6.2.1. Further frame preparation test results are graphed in section 6.2.2.



(a) Original image (b) Image blurred using Guassian kernel (c) Even rows and columns deleted (d) Result

Figure 4.1: *One step in the image pyramid generation process.*

4.2.1 Finding Image Intensity

Reducing the number of image channels from red, green and blue to a single eight-bit channel represented as "grey" may reduce the amount of data to process by a factor of three. A grey-scale image represents image *intensity*. There is no single correct method of converting a colour image to **greyscale**, but the most common means of doing so is to match the luminance of the greyscale image to the luminance of the colour image. The grey value of every pixel position is calculated by adding together 30% of the red value, 59% of the green value and 11% of the blue value at that pixel position. The equation is given by:

$$I(x, y) = 0.3 \times R(x, y) + 0.59 \times G(x, y) + 0.11 \times B(x, y)$$

From this equation we can see that the processor needs to perform three floating point multiplications and two floating point additions per pixel position for a total of 1 536 000 floating point operations. An optimised 600 MHz processor can perform 13 times as many *integer* operations per frame. Optimisations, such as bit-shifting values to work entirely in the integer domain, means that greyscale conversion requires only a fraction of the time between incoming video frames to complete.

4.2.2 Down-sampling

Another technique that may be used is image **down-sampling** to reduce the total number of pixels per image. Ideally the original image's aspect ratio is always preserved which means that the number of pixels are reduced by twice the scale factor. Each down-sampling operation results in loss of data which will reduce tracker accuracy and could influence overall robustness. An AR system designer may either decide that a slight loss of accuracy is acceptable or employ a multi-scale approach whereby full tracking is first done on a sub-sampled image and the results then refined using higher resolution versions of the image [19].

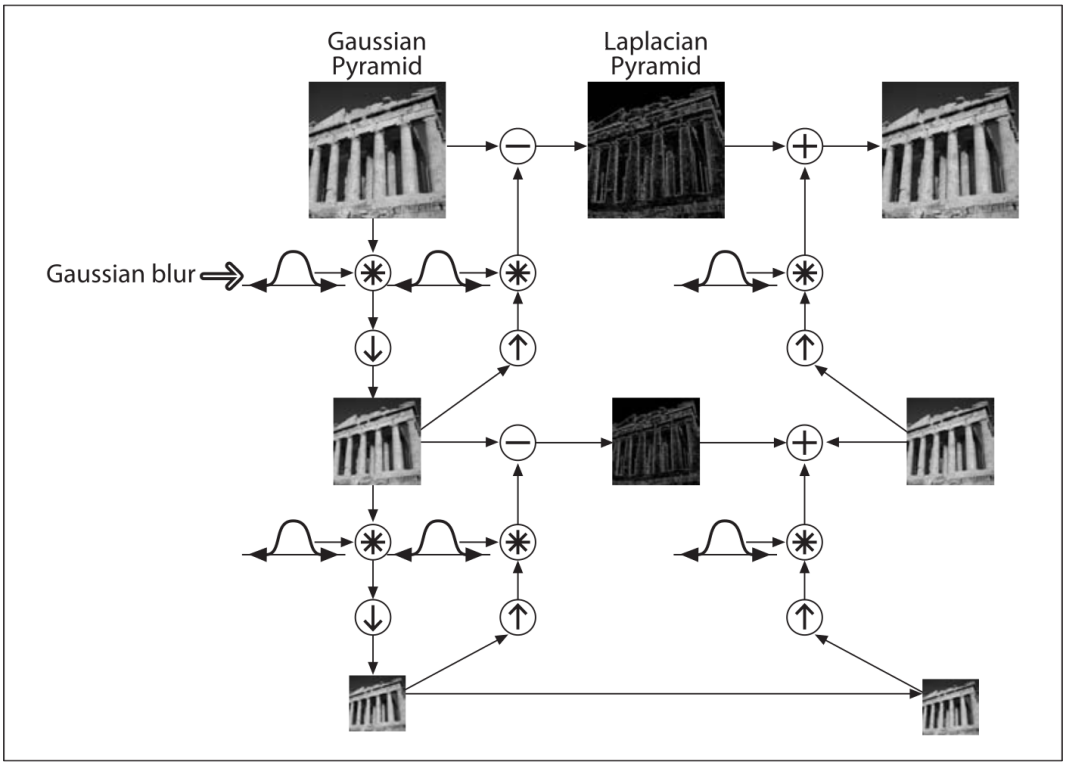
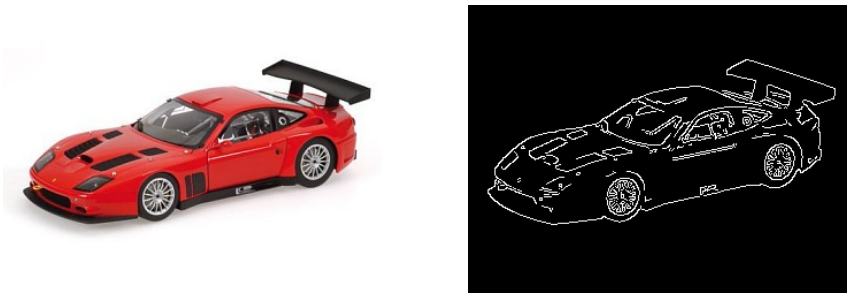


Figure 4.2: *Calculation of Gaussian image pyramid and its inverse. Picture courtesy of Bradski [6].*

In scale-space theory the concept of **Image Pyramids** are discussed by Adelson [1] and described as a set of low-pass or band-pass filtered copies of an image, each a representation of the image at a different scale. They are essentially multi-scale representations of an image starting from the original image and scaling down by a factor of two each step down to one pixel. This process can be visualised as a pyramid with the original image at the bottom.

Figure 4.2 shows the steps involved in calculating an image pyramid and the Laplacian pyramid which contains all information lost in each down-samlping step. If both the image pyramid and Laplacian pyramid are available, the original image can be perfectly reconstructed. Every level of the image pyramid contains four times fewer pixels than the layer below it which will directly affect total processing time. When constructing an Image Pyramid, down-sampling follows two steps as demonstrated in figure 4.1. First the image is slightly blurred using a 5 x 5 Gaussian kernel. This reduces the effect of image sensor noise but may be an expensive operation. Next, every even row and column of pixels are deleted. The Gaussian blur in the first step also ensures that not all the information in the even rows and columns are lost. Without the Gaussian-blur step the down-sampled images becomes "pixelated" and may introduce unnatural edges and corners which is bad for the next step in tracking, namely *feature detection.*



(a) Ferrari 575 GTC Model (b) Canny edge detection result

Figure 4.3: *Canny Edge Detection.*

4.3 Feature Detection

Even if a 640 × 480 colour frame is converted to greyscale and sub-sampled twice, the resulting number of bits per frame are still $(640 \times 480 \times 3 \times 8)/3/4/4 = 153\,600$. Although far fewer than the original number of bits, this is still too much data to track between frames as-is. The most crucial step in reducing the amount of data to process and make real-time tracking feasible is reliably detecting and matching features across consecutive frames. There are many classes of feature detectors, but the most frequently used are **edge detectors**, **corner detectors** and **blob detectors**. Feature detection encompasses a vast field of research and describing all forms of feature detectors fall outside the scope of this report. We will focus on the results and recommendations of previous works and only discuss a small subset of feature detectors to introduce the concept.

4.3.1 Edge Detection

As discussed in section 2.2, many model- and marker- based trackers makes use of edge detectors to match real-world edges with known model edges in order to determine pose. There are many edge detection algorithms, but one of the most popular and most successful is a method refined by Canny [7]. Figure 4.3 demonstrates the result of a Canny Edge detector applied to a toy car model. As we can see from this figure, the shape of a model can best be estimated if we can detect its outline and contours. The ideal edge detector will always return such a set of connected curves. Tracking contours directly related to structures within the image instead of individual pixels significantly reduces the amount of data to process which is exactly the purpose of feature detection.

The Canny algorithm performs a sequence of steps to perform edge detection:

1. The image is slightly blurred using a Gaussian filter to reduce the impact of “noisy” pixels that may be present due to imperfections in the imaging sensor.

2. The *intensity gradient* of this image is found by computing the first derivatives of the image in the x- and y- directions and the two diagonal directions.

3. A sudden change in image intensity could represent an edge. These areas are found by searching for points where the directional derivatives are local maxima.

4. Given the above points as candidates for edges, the area around each point is inspected to determine whether it actually lies on an edge and the direction of the edge. This stage is called *non-maximal suppression* and returns a set of edge points. Non-maximal suppression is discussed further in section 4.3.2.

5. Edges are traced using edge points and hysteresis thresholding. After this step, a binary image is returned where edge pixels are represented by ones and non-edge pixels represented by zeros.

The binary image returned may be further processed to find curves and polygons, but this is not part of the Canny edge detector stage. There exists other edge detection strategies, but according to Shapiro [34] the Canny edge detector and its variations are still "state-of-the-art" and performs significantly better than many other edge detector algorithms.

4.3.2 Interest Point Detection

As already stated it would be infeasible and unnecessary to track every pixel in an image from frame to frame. We would like to identify visually significant "key points" in one frame that we can reliably find again in the following. Tracking pose relies on finding enough of these point correspondences between frames. Before we can identify such points, however, we need to detect them first. The most important aspect of interest point *detection* is to do it as consistently as possible. The same interest points that are detected in one frame must be detected again in the following frame, given that they are still present on the image.

If we recorded a blank piece of paper moving in front of a white background, it would be impossible to track because the system could not distinguish between one point on the image and another. If we were to draw a single black dot on the paper, it would suddenly become very easy to follow because the system would only need to find that one dot. The dot can be seen as a discontinuity or a point where a sudden change in intensity occurs in the image. The first characteristic of an interest point is therefore being a *discontinuity*. If we were to add many more such dots on the paper it would become impossible to differentiate between them and we would have the same problem as with a blank piece of paper. We can only distinguish between them if the local region surrounding one differs at least to some degree from the local region surrounding all others. The second characteristic of an interest point is therefore that it and the localised region surrounding it must be *unique*.

Harris Corners

The most obvious example of a point of interest is a corner, or the intersection of two edges. A number of *corner detection* algorithms have been developed over the years. The majority of interest point detection algorithms compute an interest point response function, C, for

points across an image (either all pixels, or only some selected pixels). Probably the most famous corner detector is known as the "Harris corner detector" developed by Harris [11] which was created for the 3-D interpretation of image sequences. As with the Canny edge detector, calculating the second-order partial derivative of an image in a specific direction will indicate regions of sharp changes in image intensity (discontinuities) in that direction. Given a two dimensional greyscale image, I, so that the image intensity at a specific pixel is given by $I(x, y)$, Harris determines C_{Harris} for a specific point as follows:

First, we take an image "patch" over an area (u,v) and shift it by (x,y). The shape of the patch can be circular or rectangular. The weighted *sum of squared differences* (SSD) between the original patch and the shifted patch is denoted by S and given by:

$$S(x, y) = \sum_u \sum_v w(u, v) \ (I(u + x, v + y) - I(u, v))^2 \tag{4.1}$$

$w(u, v)$ is the weighting term and can be uniform, but is often used to create a circular or Gaussian weighting.

$I(u+x, v+y)$ can be approximated by a Taylor expansion. With I_x and I_y the first-order partial derivatives of I in the horizontal and vertical directions, we have:

$$I(u + x, v + y) \approx I(u, v) + I_x(u, v)x + I_y(u, v)y \tag{4.2}$$

Substituting (4.2) back into (4.1), $S(x, y)$ can be approximated as:

$$S(x, y) \approx \sum_u \sum_v w(u, v) \ (I_x(u, v)x + I_y(u, v)y)^2 \tag{4.3}$$

This can be written in matrix form:

$$S(x, y) \approx \begin{pmatrix} x & y \end{pmatrix} \mathbf{H}_{Harris} \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.4}$$

where $\mathbf{H}_{Harris}$ is the structure tensor,

$$\mathbf{H}_{Harris} = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2(u, v) & I_x(u, v)I_y(u, v) \\ I_x(u, v)I_y(u, v) & I_y^2(u, v) \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \tag{4.5}$$

The angle brackets denote averaging (summation over (u,v)).

$\mathbf{H}_{Harris}$ is known as the *Harris Matrix*. A point of interest is characterized by a large variation of S in both the horizontal and vertical directions. This can be determined by examining the two eigenvalues of $\mathbf{H}_{Harris}$, namely λ_1 and λ_2. Harris determined that a "good" point of interest would have two large eigenvalues, however, calculation of the eigenvalues may be computationally expensive and so Harris defined the corner response (C_{Harris}) as:

$$C_{Harris} = \lambda_1\lambda_2 - \kappa \, (\lambda_1 + \lambda_2)^2 = \det(\mathbf{H}_{Harris}) - \kappa \ \mathrm{trace}^2(\mathbf{H}_{Harris}) \tag{4.6}$$

C_{Harris} is large if both eigenvalues are large and avoids explicit computation of the eigenvalues. The value of κ was determined empirically as ranging from 0.04 to 0.15.

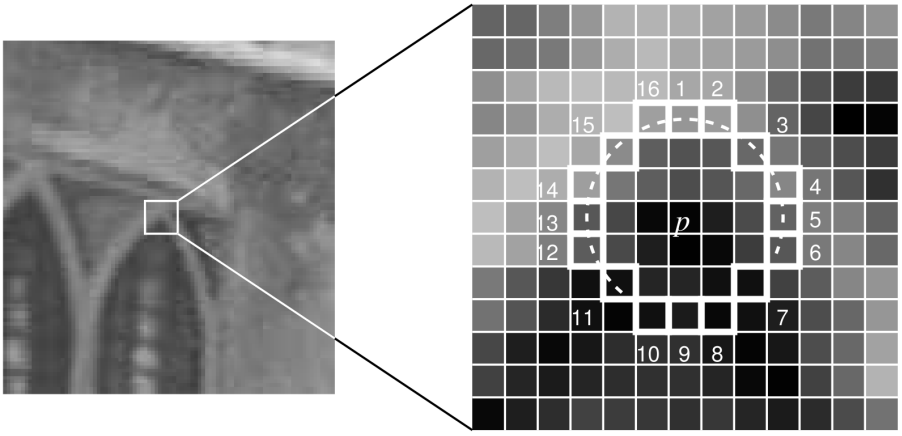


Figure 4.4: *FAST segment test circle around a pixel p. Picture courtesy of [31].*

Shi-Tomasi corners

Shi and Tomasi [35] later improved upon the Harris detector and found that good corners resulted as long as the smallest eigenvalue was greater than a certain threshold. They determined that $C_{Shitomasi} = \min(\lambda_1, \lambda_2)$ is a better measure of a good corner than C_{Harris}. Unfortunately this does require explicit calculation of the eigenvalues, which may be computationally expensive. Eigenvalues of a 2-by-2 matrix are calculated as follows:

$$\lambda_{1,2} = \frac{1}{2}\left(\text{trace}(\mathbf{H}_{Harris}) \pm \sqrt{\text{trace}^2(\mathbf{H}_{Harris}) - 4\det(\mathbf{H}_{Harris})}\right) \tag{4.7}$$

Calculation of the square root may negatively affect computation time. In section 6.3 we show the result of a number of tests comparing Harris, Shi-Tomasi and FAST corner detectors. The results show that Shi-Tomasi does indeed have a better criteria for an interest point response function, returning more points than Harris for the same threshold. The results also show that Shi-Tomasi is slower than Harris for the same threshold, possibly due to the calculation of eigenvalues.

The de-facto corner detector used with the popular computer vision library "OpenCV" is known as `cvGoodFeaturesToTrack()` and uses this implementation by Shi and Tomasi. This method is sometimes also referred to as the "Kanade-Tomasi" corner detector [35].

FAST corners

The *Features from Accelerated Segment Test* (FAST) corner detection algorithm presented by Rosten and Drummond [32, 31] is a key aspect of this study, without which our prototype application would likely not have existed. In stark contrast with the costly interest point detection methods such as Harris, FAST takes a far more qualitative approach to determine whether a pixel represents a corner. For every pixel, p, in a greyscale image, FAST examines 16 pixels in a circle of radius three around p as shown in figure 4.4. The intensity (grey value) of a pixel is given by $I(x, y)$. FAST simply states that p can be classified as an interest point

Figure 4.5: *FAST corner detector.*

if the intensities of at least 12 contiguous points on the test circle are all brighter or all darker than $I(x_p, y_p)$ (the intensity of p) by some threshold, t. An interest point can be categorised as "positive" (or bright) if at least 12 contiguous circle points have intensities greater than $I(x_p, y_p) + t$, or "negative" (dark) if their intensities are smaller than $I(x_p, y_p) - t$. This partitioning can be useful as positive points do not have to be compared to negative points in later stages of tracking.

Since the 12 points must be contiguous (connected), the test can be accelerated by first examining only pixels 1, 9, 5 and 13 (North, West, South, East). An interest point can only exist if three of these test points are all brighter or all darker than the intensity of p by the threshold. If more than one of these test points fall inside the threshold range, p can be rejected immediately. If this initial test passes, the remaining pixels on the circle are tested. Rosten found that due to this optimisation, the algorithm examines on average only 3.8 pixels on the test circle to test whether a given point is a point of interest.

Figure 4.5 shows the FAST corner detector applied to an image. For generating this image we used a FAST threshold of 50.

This description of FAST should immediately indicate two major potential advantages of FAST over other interest point detectors:

1. **FAST is fast!** All that is required to process an entire image is an average of 3.8 *integer* additions and comparisons per pixel. In comparison to other detectors that

require the calculation of partial derivatives, square roots and convolutions, FAST seems much less computationally complex. In section 6.3.2 we show the results of a number of tests performed on an iPhone 3Gs to determine the speed of FAST.

2. **Simple thresholding**: The threshold, t, is an integer value ranging between 0 and 255 for an 8-bit greyscale image. Adjusting t will directly influence the number, and possibly quality, of interest points detected. For applications where a constant number of interest points need to be detected, t can easily be dynamically adjusted.

An important question that must be answered is whether FAST is *reliable*. In section 6.3.3 we perform a series of tests on an iPhone 3Gs to determine the consistency of FAST. The detection method used by FAST is very simple compared to mathematically rigorous methods such as Harris and Shi-Tomasi. Even the authors of FAST were sceptical about their method at first. In "Machine learning for high-speed corner detection" [31] they perform a series of tests comparing the ability of FAST to detect consistent points across images to a number of other detectors. The test criterion was that the same scene, viewed from two different angles, should yield features which correspond to the same 3-D locations. In their report they write: "contrary to our initial expectations, we show that despite being principally constructed for speed, our detector significantly outperforms existing feature detectors according to this criterion". They do note that there are a number of **disadvantages** to FAST:

1. FAST performs poorly on images with only large-scale features such as blurred images. Most other corner-based interest point detectors also suffer from this disadvantage.

2. FAST is not very robust to noise in an image. Since its high speed is achieved by examining the fewest amount of pixels possible per point, it does not have the ability to average out noise like other detectors that examine Gaussian weighted patches.

3. Multiple points of interest may be detected next to each other.

The third disadvantage can be problematic, especially if FAST is used to detect points of interest that will be tracked. FAST may detect multiple "points of interest" around a single corner in an image if the corner is larger than one or two pixels. It is generally preferable to track points that are spaced at least a small distance apart so that their regions used for feature description not overlap. FAST makes provision for this requirement by offering an optional step after an interest point candidate is found known as *non-maximal suppression*. Non-maximal suppression works by calculating a score function, V, for each interest point candidate by summing the difference in intensity between $I(x_p, y_p)$ and each of the contiguous pixels in the test circle that are all brighter or darker than $I(x_p, y_p)$ by the threshold t (those pixels that passed the 12-point test). With non-maximal suppression enabled, only the candidate with the highest score in a local region is accepted as a point of interest. Rosten [31] gives the following definition for a point score function, V:

$$V = \max\Big(\sum_{u \in S_{bright}} |I(x_u, y_u) - I(x_p, y_p)| - t \ , \sum_{u \in S_{dark}} |I(x_p, y_p) - I(x_u, y_u)| - t\Big)$$

where S_{bright} and S_{dark} are the sets of contiguous pixels around p that are either all brighter or darker than p by the threshold t. They are defined as:

$$S_{bright} = \{u | I(x_u, y_u) >= I(x_p, y_p) + t\}$$

$$S_{dark} = \{u | I(x_u, y_u) <= I(x_p, y_p) - t\}$$

Non-maximal suppression may incur a small performance penalty in the detection stage, but it can lead to large savings in computational requirements in the feature description and matching stage. In section 6.3 we compare FAST to other corner detectors and show that it is considerably faster and consistent enough to be used for our AR prototype.

4.4 Feature Description

As discussed in chapter 3, finding point correspondences is vital to successful triangulation and calculating camera pose. In order to find corresponding features over video frames it must be possible to detect features as described in the previous section, but it is also important to *identify* features and match them between frames. We call this *feature description* and it involves "extracting" feature information. In the previous section, section 4.3, we have defined a good feature as being *unique* so that it is easily distinguishable from others.

As with feature detection, a wide variety of feature description algorithms have been presented over the years. A good feature descriptor must exhibit these three characteristics:

1. **Repeatability**: The feature descriptor should be reliable, finding the same physical interest points under different viewing conditions. It must have high accuracy and a low false-positive rate. It should be invariant to changes in rotation, translation and scale.

2. **Robustness**: The feature descriptor must be able to identify the same point between frames even if there are changes in illumination, changes in noise and small changes of the viewpoint.

3. **Speed**: It must be able to extract feature information and match it against a large database as quickly as possible, preferably in real-time.

Lowe [24] introduces a method for image feature extraction called the Scale Invariant Feature Transform (**SIFT**). It is invariant to image scaling, translation and rotation as well as being at least partially invariant to changes in illumination and 3-D projective transforms. This approach transforms an image into a large collection of local feature vectors called "SIFT keys" that are used for identification. Wagner [38] uses a "heavily modified" version of SIFT to enable marker-based natural feature tracking on mobile phones in real-time. Wagner also mentions the feature descriptor "SURF", noting that it is potentially more efficient but still do not attain real-time performance in its unmodified form.

4.4.1 SURF

Speeded-Up Robust Features (**SURF**) are introduced by Bay *et al* [4][1] and is similar to SIFT. According to the authors of SURF, it outperforms SIFT and other "previously proposed schemes" in almost all cases with respect to repeatability, distinctiveness, robustness and speed. It is scale-, rotation- and translation-invariant. Similar to SIFT's "SIFT keys", SURF calculates a feature vector for the neighbourhood of every interest point. Matching is based on the difference between vectors. Because Bay *et al* shows that SURF outperforms SIFT, we will not include an explanation of SIFT but we will provide a brief description of SURF.

SURF includes an interest point detection step that makes use of a Hessian matrix-based approach. A Hessian matrix is very similar to the Harris [11] matrix, $\mathbf{H}_{Harris}$, discussed in section 4.3.2 as it also relies on the calculation of the second order derivatives of an image in two directions. Bay *et al* adapted a Hessian matrix-based approach for multi-scale interest point detection. Because SURF's built-in interest point detector can be replaced with the much faster FAST detector, we will not discuss the theory of the interest point detection step. It is discussed in detail in the SURF paper by Bay *et al* [4].

The SURF **descriptor** describes the distribution of intensity (greyscale) content within the neighbourhood of a given interest point. Bay *et al* mentions that the gradient information that is extracted is similar to the information extracted by SIFT and its variants. SURF calculates the distribution of first order *Haar wavelet* responses in the x- and y- directions in an image instead of calculating the image gradient. To increase performance, SURF uses integral images and only 64 fields per feature described. The integral image, $I_{\sum}$, of an image, I, is given as:

$$I_{\sum}(x,y) = \sum_{i=0}^{i<=x} \sum_{j=0}^{j<=y} I(i,j)$$

where $I_{\sum}(x,y)$ is the value of an integral image pixel at a point (x,y) and $I(i,j)$ the value of a pixel in a greyscale image at a point (i,j). Calculating the integral image allows for the calculation of the sum of intensities over a rectangular area of any size in an image using only three additions. SURF uses big filter sizes that regularly need to find the sum of intensities inside rectangular areas in an image, and so it benefits from the calculation of the integral image.

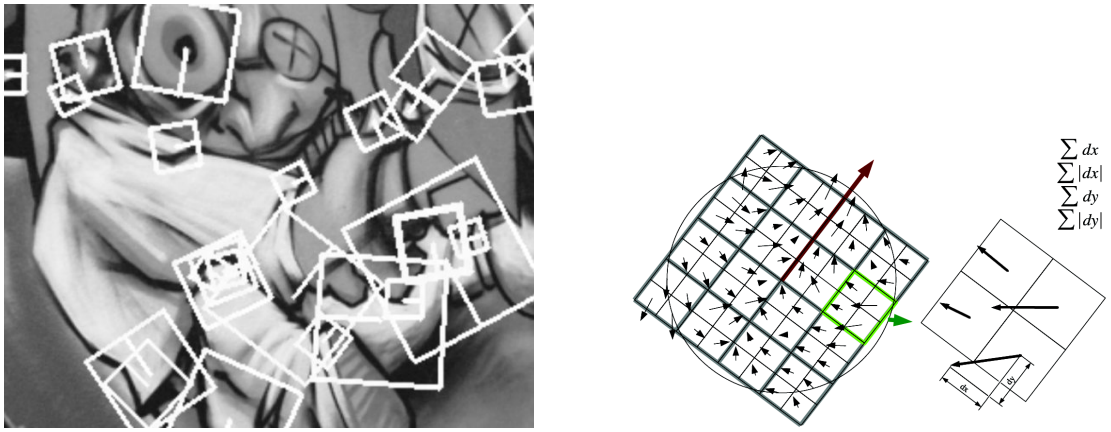
SURF performs the following steps to calculate feature descriptors:

1. Given an image, I, the integral image, $I_{\sum}$, is calculated in O(n) time.

2. Interest points are detected over several scales using a Hessian-matrix based multi-scale detector.

---

[1] We were fortunate enough to work under Luc van Gool, co-author of SURF, while working on the Aerial Crowds project as mentioned in section 2.2.2.



(a) SURF interest point orientation estimation. (b) SURF 64 byte descriptor

Figure 4.6: *Calculation of SURF descriptors. Pictures courtesy of [4].*

3. For each interest point, a reproducible *orientation* is assigned to the interest point by calculating the Haar wavelet responses in the x- and y- directions of the image within a circular neighbourhood around the interest point. The circular neighbourhood has a radius of 6 pixels multiplied by the scale at which the interest point was detected. The integral image is used for fast filtering. Figure 4.6a shows the result of the interest point orientation step.

4. The orientated squares centred on multi-scale interest points, as shown in figure 4.6a, are broken up into smaller 4×4 squares known as sub-regions as shown in figure 4.6b. For each sub-region a Haar wavelet response is calculated in the x- and y- directions relative to the orientation of the interest point. The sum of the Haar wavelet responses for a sub-region in the x- and y- directions are given by: $\sum dx$ and $\sum dy$. The sum of the two wavelet responses and their absolute values form a 2×2 matrix per sub-region. The 2×2 matrix of each sub-region correspond to the actual fields of the descriptor for a total of $2 \times 2 \times 4 \times 4 = 64$ fields as shown in figure 4.6b.

Bay *et al* does not elaborate on how descriptors are matched but we assume that they use a form of indexed data structure of descriptors that allows fast classification of descriptors to be matched within the data structure. Classification is based on the difference between descriptor vectors.

Figure 4.7 shows the result of points tracked between two images using SURF descriptors. In sections 6.4.1 and 6.4.2 we perform a series of tests on an iPhone 3Gs to determine the processing requirements, speed and consistency of SURF.

For a complete description of SURF and comparison with SIFT, please see the original paper "Speeded-Up Robust Features" [4]. An open-source implementation is available as "OpenSURF"[2]. We evaluate this implementation of SURF in section 6.4.1.

[2] `http://code.google.com/p/opensurf1/`

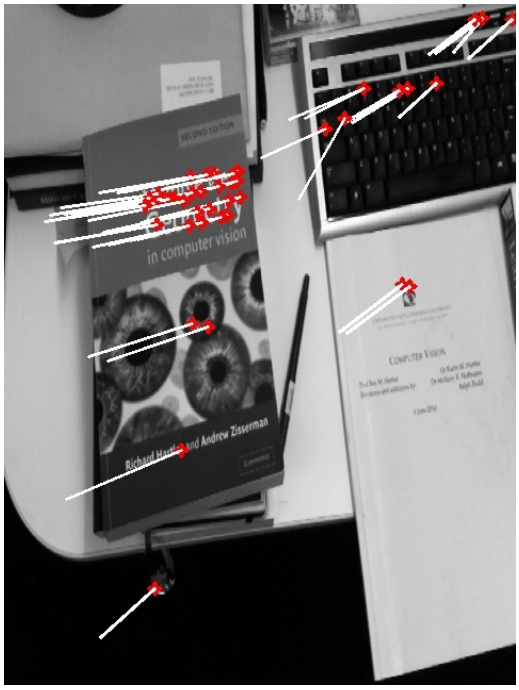


Figure 4.7: *SURF descriptor tracking.*

Initially we planned on basing our tracking strategy on SURF descriptors, but found that we could not obtain real-time frame-rates as shown in section 6.4.1 and had to find an alternative. We investigated the use of optical flow tracking instead of feature descriptor matching. Optical flow is discussed in the following section.

4.5 Optical Flow

Optical flow is a means of determining the net difference in *motion* between two frames. *Dense optical flow* is used to determine the net distance and direction that each pixel has moved between two frames. This is analogous to tracking each and every pixel in a frame, something we have shown to be unfavourable for real-time systems in section 4.2, specifically due to its high computational cost. *Sparse optical flow* is an alternative option that tracks only a small subset of pixels (or rather points) and therefore rely on an interest point detector. The same features that are good for feature description are good for optical flow tracking.

Blesser [5] writes that optical flow tracking can be used as a lightweight alternative to SLAM. It is a substitute for feature description and triangulation. The paper shows that with enough measurements, a sparse optical flow tracker can be implemented that allows tracking for extended periods of time without 3-D point registrations. We follow this advice in the development of our AR prototype application as discussed in section 5.5.3. Section 6.4.3 and section 6.4.4 shows the results of a series of timing and consistency tests performed on an iPhone 3Gs using `cvCalcOpticalFlowPyrLK`.

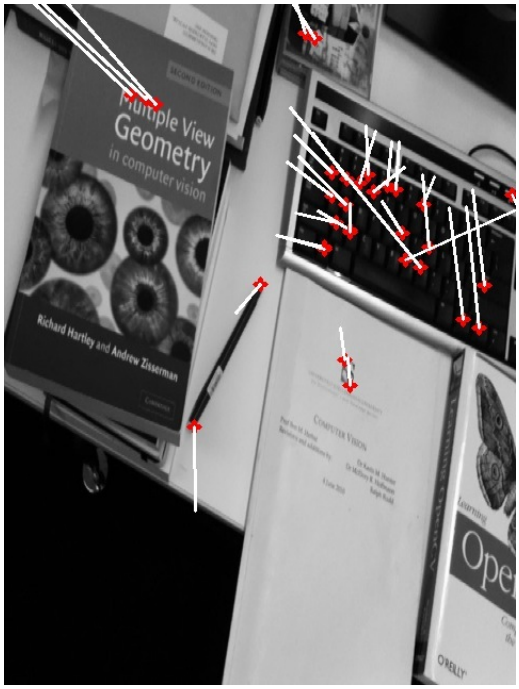


Figure 4.8: *Sparse optical flow tracking.*

4.5.1 Lucas-Kanade Sparse Optical Flow

Lucas and Kanade [25] presented an algorithm almost 30 years ago that is still being used today in the popular computer vision library OpenCV as `cvCalcOpticalFlowPyrLK` . Lucas and Kanade attempted to produce a method for dense optical flow, but their method was easily adapted to tracking only a subset of points in an input image and is suitable for sparse optical flow. The technique relies on tracking points inside a small window around the original position of the point. The disadvantage of this is that using small windows can make it very difficult for Lucas-Kanade to track large motions. To allow tracking of large motions, a multi-scale approach was developed [2] that started tracking from the lowest detail level in an image pyramid and refining the estimate by working down through the pyramid levels of increasing resolution. Image pyramids are discussed in section 4.2.

According to "Learning OpenCV" [6], the Lucas-Kanade algorithm relies on three assumptions:

1. The **intensity** (section 4.2.1) of a point does not change much from frame to frame.
2. A point only moves **small distances** from frame to frame.
3. Neighbouring points have **similar motion** from frame to frame.

For simplification we discuss optical flow in one dimension (1-D) first and will then expand to optical flow in two dimensions (2-D). The first requirement, intensity consistency, says that a pixel, p', on a one-dimensional image at position $x(i)$ at time i will have the same

intensity in a following frame at position $x(i + di)$ a time-step di later. We define $f(x, i)$ as the intensity of a point at position $x(i)$ at time i. From the first requirement we have:

$$f(x, i) \equiv I(x(i), i) = I(x(i + di), i + di) \tag{4.8}$$

Where $I(x(i), i)$ is the intensity of a pixel on a one-dimensional image at position $x(i)$ at time i.

Since the intensity of the pixel does not change over time, we can write:

$$\frac{\partial f(x)}{\partial i} = 0 \tag{4.9}$$

Substituting (4.8) into (4.9) we have:

$$\frac{\partial I(x(i), i)}{\partial i} = 0 \tag{4.10}$$

Applying the chain rule for partial differentiation:

$$\frac{\partial I}{\partial x}|_i \left(\frac{\partial x}{\partial i}\right) + \frac{\partial I}{\partial i}|_{x(i)} = 0 \tag{4.11}$$

$I_x = \frac{\partial I}{\partial x}|_i$ is the partial derivative of the image, I, with respect to x evaluated at time $t{=}i$, $I_i = \frac{\partial I}{\partial i}|_{x(i)}$ is the partial derivative of I with respect to i and $u = \frac{\partial x}{\partial i}$ is the *velocity* or relative movement of a point in one dimension.

Substituting I_x, I_i and u into (4.11):

$$I_x u + I_i = 0 \tag{4.12}$$

We see that we need to calculate a spatial intensity derivative for an initial image and a temporal intensity derivative between two frames in order to find u.

According to Bradski [6] we can generalise equation (4.12) for two dimensions as follows:

$$I_x u + I_y v + I_i = 0 \tag{4.13}$$

Where I_y is the partial derivative of an image, I, in the y-direction and v is the relative movement of a point in the y-direction. We can rewrite this equation as:

$$I_x u + I_y v = -I_i \tag{4.14}$$

This equation is related to only a single pixel, but unfortunately there are two unknowns (u and v) so it can not be solved alone. In order to find the movement of a point, we look at the third assumption that the Lucas-Kanade algorithm relies on, namely that points in the neighbourhood around one point will generally move in the same way. We can use the surrounding points to set up a system of equations. If we use a 3-by-3 window of "helper" points around a point, we can set up 9 equations:

$$
\begin{aligned}
I_x(p'_1)u + I_y(p'_1)v &= -I_i(p'_1) \\
I_x(p'_2)u + I_y(p'_2)v &= -I_i(p'_2) \\
&\vdots \\
I_x(p'_9)u + I_y(p'_9)v &= -I_i(p'_9)
\end{aligned} \tag{4.15}
$$

We can rewrite equation (4.15) in matrix form:

$$
\begin{bmatrix} I_x(p'_1) & I_y(p'_1) \\ I_x(p'_2) & I_y(p'_2) \\ \vdots & \vdots \\ I_x(p'_9) & I_y(p'_9) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_i(p'_1) \\ I_i(p'_2) \\ \vdots \\ I_i(p'_9) \end{bmatrix} \implies \mathbf{A}\vec{d} = -\vec{c} \tag{4.16}
$$

where:

$$
\vec{c} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.17}
$$

According to Bradski, the system can be solved by least-squares minimisation of $\mathbf{A}\vec{d} - \vec{c}$ in standard form as:

$$
(\mathbf{A}^T\mathbf{A})\vec{d} = \mathbf{A}^T\vec{c} \tag{4.18}
$$

This can be rewritten as:

$$
\begin{bmatrix} \vec{u} \\ \vec{v} \end{bmatrix} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b} \tag{4.19}
$$

According to Bradski, this is can be solved as long as the image region inside our window has changing texture in at least two directions. Because of the second assumption, that a point only moves small distances between frames, `cvCalcOpticalFlowPyrLK` tracks over courser scales of an image pyramid first and then refines its initial motion estimate by working down through the levels of the image pyramid and searching within the window calculated from the previous step. Figure 4.8 is an example of the OpenCV implementation of the multi-scale Lucas-Kanade tracker, `cvCalcOpticalFlowPyrLK`, applied to two pictures of the same scene taken from two angles.

This OpenCV method can be optimised by given it a set of *predicted* motions of points in the form of an array indicating their predicted new positions. Point movement can be modelled and predicted by using an Extended Kalman Filter (EKF) [5]. Using an EKF falls outside the scope of this study and is left as future work as discussed in chapter 7.

The method can be further optimised by keeping track of the image pyramids calculated for each frame. Each iteration, at least two image pyramids are required. One for the previous frame's image $(t-1)$ and one for the current frame's image (t). If we store the image pyramid calculated for the current video frame, we may give it to the function in the following iteration so that it does not have to be recalculated.

4.6 Pose Estimation

The theory behind estimating pose, specifically the rotation matrix $\mathbf{R}$ and translation vector $\vec{r}$ is discussed in detail in chapter 3 specifically sections 3.4 and 3.5. As we do not use full computer vision pose estimation in this thesis we will not cover pose estimation in depth in this section. "Computer Vision with the OpenCV Library" [6] and "Multiple View Geometry in Computer Vision" [13] contain in depth descriptions of computer vision based full pose estimation.

4.7 Summary

In this chapter we provided an overview of specific computer vision techniques used in computer vision based tracking systems. In the following chapter we will describe our implementation of some of these techniques in our own prototype application.

Chapter 5

Mobile AR System Design

5.1 Introduction

In this chapter we will discuss the design of a prototype markerless mobile augmented reality system that utilises integrated sensor data to facilitate pose estimation. The use of a markerless system on a smart-phone platform for development has been motivated in section 1.4. First we will look at the requirements of such a system and then discuss the choice of hardware components, visual tracking methods, integration of sensor data and visual output.

5.2 System Requirements

The choice of platform and tracking type has already been discussed and motivated. In terms of technical specifications, the most obvious requirement is that the system must run in real-time as per the definition of AR discussed in section 1.1. For this thesis we assume that the minimum frame-rate that can represent real-time on a mobile device is 10 frames per second. This gives us a *frame budget* of $\frac{1}{10} = 0.1s$ or 100 milliseconds. The remaining two technical specifications that are slightly more difficult to measure are accuracy and robustness.

Accuracy is defined as the ability to track corresponding points between consecutive frames. If a system is not able to track enough points accurately, it will suffer from "drift" where the Augmented Reality view appear to drift on top of the real world. We aim to minimise drift and maximise accuracy.

Robustness is defined as accurate tracking of corresponding points between two frames, even with the presence of noise, large changes in scale, large translations, large rotation and large changes in illumination. Designing a robust system requires in-depth knowledge of advanced computer vision techniques and tracking strategies. We will not attempt to design a system that is highly robust, but will expect our system to perform reasonably well (accurate registration, minimal drift) even in the presence of low amounts of noise introduced by an imperfect camera, small changes in scale, small translations and small rotations between frames caused by gradual camera movement.

In terms of subjective specifications, the system must at least be visually demonstrateable.

We will define the system specifications as:

- The system must be implemented on a smart-phone device.
- The system must be fully markerless.
- The system must make use of an integrated accelerometer sensor to estimate orientation ($\mathbf{R}$).
- The system must perform at real-time frame rates of at least 10 frames per second. Tests determining the speed of the system is presented in section 6.5.1.
- Pose estimation must be reasonably accurate as defined above (minimal drift given gradual camera movement). Tests determining the accuracy of the system is presented in section 6.5.2.
- The system must be visually demonstrateable.

5.3 Device Selection

For the purpose of this study a number of mobile smart-phone devices were evaluated as potential development platform. At the time of evaluation we felt that the most comfortable development environments were offered by the Apple iPhone and Android SDKs. Both offered familiar development environments - high-level, object oriented and with well defined APIs - and OpenCV can be compiled on both platforms. In this section we evaluate a number of Apple and Android devices. An Apple iPhone 3G was generously provided by a sponsor, Antonie Roux, and was later replaced by an Apple iPhone 3Gs by the same sponsor when it became available. From the comparisons below we found that both the Apple iPhone and Android development environments are good platforms for AR development.

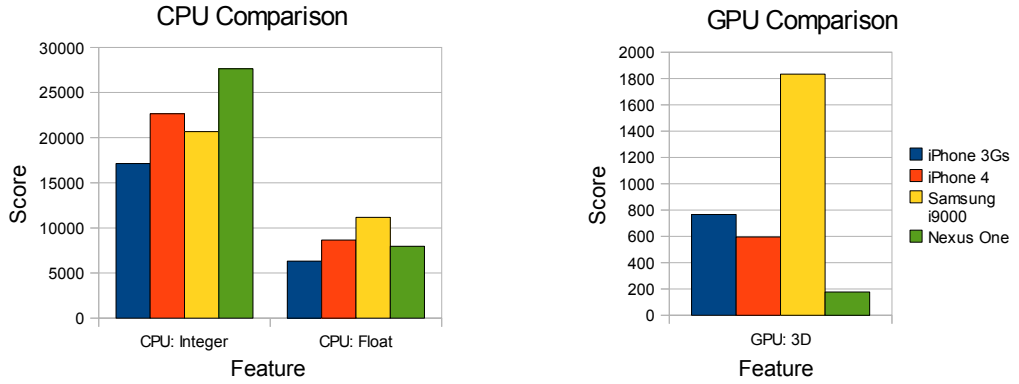
Table 5.1 places the 3Gs next to three other "high-end" smart-phones released in 2010 in order to show that the 3Gs is a good representative of the current smart-phone market according to specifications and features. A performance comparison of the four devices is shown in figure 5.1 in order to place the 3Gs in terms of processing power and 3-D capabilities.

In table 5.1, it is shown that the system specifications are fairly similar. The central processing units of the iPhone 3Gs, iPhone 4 and Samsung i9000 are based on the ARM Cortex-A8 core and the Nexus One's CPU is very similar to the Cortex-A8. Even though the 3Gs has only half the memory of the other devices, it is still more than enough for an AR application that does not feature large-scale mapping of the environment or other large datasets.

The 3Gs falls slightly short when it comes to camera quality and screen resolution, although this does not pose a significant problem as incoming video frames are usually down-sampled due to processing power constraints as discussed in section 4.2.

2010 Smart-phones				
	iPhone 3GS	**iPhone 4**	**Samsung i9000**	**Nexus One**
				
CPU	ARM Cortex-A8 @ 600 Mhz	Apple A4 @ 700 - 1000 Mhz (undetermined)	Samsung Hummingbird @ 1000 Mhz	Qualcomm Snapdragon @ 1000 Mhz
GPU	PowerVR SGX530	PowerVR SGX535	PowerVR SGX540	Adreno 200
RAM	256 MB	512 MB	512 MB	512 MB
Screen Resolution	320x480 (3.5”)	640x960 (3.5”)	480x800 (4”)	480x800 (3.7”)
Camera Resolution	640x480 @ 30 fps	1280x720 @ 30 fps	1280x720 @ 30 fps	720x480 @ 20+ fps
3-Axis Accelerometer	✓	✓	✓	✓
3-Axis Gyroscope	✗	✓	✗	✗
Digital Compass	✓	✓	✓	✓

Table 5.1: *Comparison of four High-end Smart-phones as of 2010*



(a) Processor Integer and Floating Point Operations Comparison

(b) GPU 3-D Skinning Operations Comparison

Figure 5.1: *Device Performance Characteristics Comparison Charts*

The integrated sensors on all devices are the same, except for the iPhone 4 which includes a 3-Axis gyroscope. The addition of a digital gyroscope to mobile devices will greatly aid the development of AR applications in the near future.

Figure 5.1 shows a comparison of the performance scores of each device as recorded at `http://www.glbenchmark.com`. The key for the chart on the right is the same for the image on the left. The CPU performance of the 3Gs is lower than the other models which indicates that AR applications deployed to newer devices will benefit from the increased processing power.

It is interesting to note that the GPU performance of the 3Gs seems better than the iPhone 4. This could possibly be attributed to the difference in resolutions with the iPhone 4 having four times the number of pixels on screen compared to the 3Gs. From this comparison we see that the 3Gs compares well to the latest smart-phone devices, but that ever increasing performance will lead to more authentic AR experiences in the future.

5.4 Process Flow

In this section we discuss the process flow of a typical AR system. After initialisation, the system typically enters a run-loop that cycles through a series of tasks like handling user input, updating the system state and rendering until the user signals that the system must terminate. Our system has four main tasks:

- **Video frame acquisition** (Approximately 15 Hz): The camera hardware signals that a video frame is available and executes a subroutine that copies the latest video frame to a frame buffer. This happens approximately every 35 milliseconds and takes between 0.5 and 11 milliseconds to complete as shown in the tests in section 6.2.1.

- **Accelerometer data acquisition** (100 Hz): This task is manually scheduled to execute every 10 milliseconds and store the accelerometer data in a data buffer. Reading data from the sensor is practically instantaneous, completing in less than 1 ms.

- **Rendering** (30-60 Hz): A new frame is rendered and drawn every 33 to 15 milliseconds and completes almost instantaneously (under 1 ms). The frequency of renders depends on the overall state of the system and is not constant, although the minimum frame rate will be above 30 (real-time).

- **Frame processing** (Approximately 10 Hz): This task processes consecutive video frames and updates pose by using the computer vision techniques discussed in section 5.5. Unlike the other tasks that are scheduled and take only a very small amount of time to execute, frame processing may take up to 100 milliseconds to complete and a new frame processing task is started as soon as the previous one has finished. The result of a number of tests determining frame processing time is shown in section 6.5.

Processing occurs on two threads. Because frame acquisition, accelerometer data acquisition and rendering all complete very quickly (typically less than 1 ms), and can execute independently of each other, they are all scheduled with varying frequencies on the **main thread**. The effect is that these tasks are *interleaved* on the main thread. Tasks on the same thread can not execute concurrently. This means that a task that takes a long time to complete such as *frame processing* would "block" other scheduled tasks and cause user interface (UI) lag if it was scheduled on the main thread. In order to ensure smooth output, frame processing is scheduled on a **background thread** and can execute concurrently to the three other tasks. This is shown in figure 5.2.

Figure 5.2 shows a visual representation of the system's process flow. Solid arrows indicate process flow and dashed arrows indicate a task requesting data from a buffer. The frequency at which each task is scheduled is indicated along its loop-back arrow. As soon as system initialisation is complete, all four tasks are scheduled to start at once and to repeat at various frequencies independently of each other. By using **buffers** namely the *frame buffer*, *accelerometer data buffer* and *pose estimate buffer*, tasks can retrieve data as necessary and are never blocked by another task. Because of this, there is a slight delay of approximately 100 milliseconds after rendering starts that tracking does not yet occur. After 100 milliseconds all buffers contain data and the system operates normally. Buffers are locked while they are being written to, so any task requesting data from a buffer in a state of update will simply wait until the buffer is free. We took care to ensure no race conditions occur by locking shared variables so that only one thread has access to a shared variable at any time. This prevents writes to buffers that are being read from. The pose estimate buffer stores the current translation and scale of the pose, as well as the rotation estimate from the accelerometer data after it is calculated as discussed in section 3.6.1.

The video feed and rendered graphics need to be combined in order to represent an AR view. This is done automatically by configuring a view to display the raw video feed

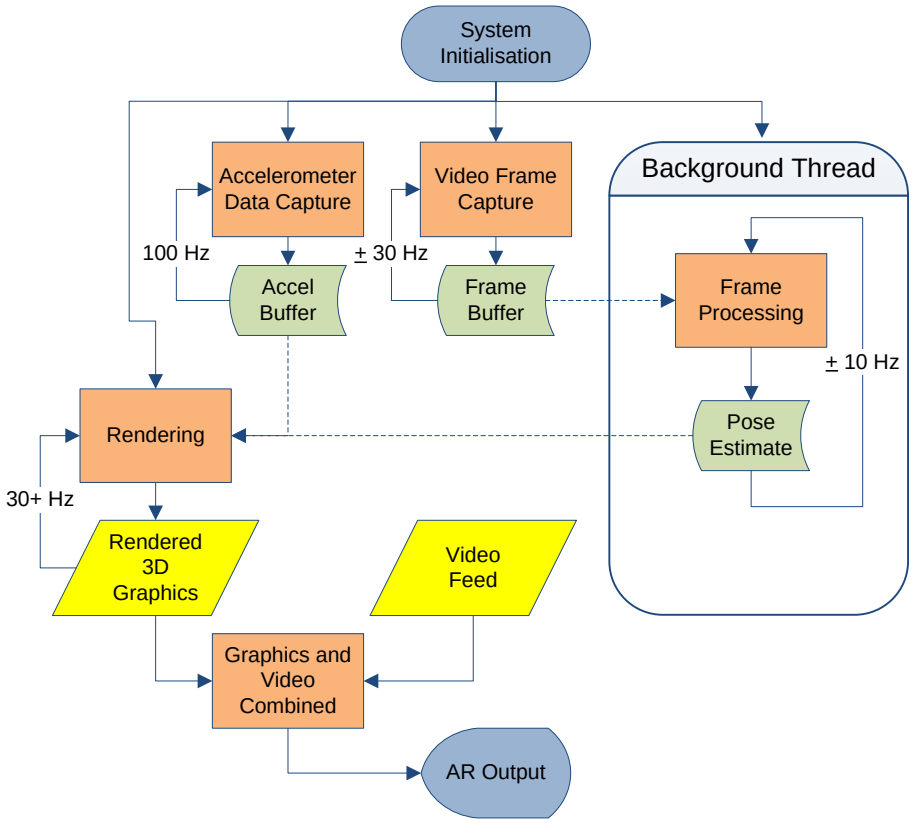


Figure 5.2: *System Process Flow. Solid arrows indicate process flow and dashed arrows indicate a task requesting data from a buffer.*

coming from the camera and overlaying the OpenGL rendered graphics with a transparent background on top of the camera view.

5.5 Visual Tracking

Visual tracking is implemented in the *frame processing* task discussed in the previous section.

5.5.1 Pre-processing

The iPhone 3Gs and iPhone 4 allows for the configuration of a "capture preset" that determines the quality of incoming video frames from the camera sensor. The camera of the iPhone 3Gs and iPhone 4 can be configured in one of three modes: High-quality, medium-quality or low-quality. The 3Gs returns video frames at resolutions of 640 x 480, 480 x 360 and 192 x 144 depending on camera mode. The results of a number of tests measuring the frame preparation time of each camera mode is shown in the tests presented in section 6.2.2.

As soon as a video frame is acquired it is prepared by converting it to greyscale. According to the tests in section 6.2.2, for the low-quality camera mode this takes approximately 2 ms, or 2% of the frame budget. At first we chose the medium-quality camera setting and implemented a multi-scale image pyramid based approach as discussed in section 4.2. We found that this consumed a considerable amount of the frame budget as shown later in section 6.2.1. Instead, we chose to configure the camera in low-quality mode to return an image of the lowest resolution for further processing. Unfortunately this reduces the quality (resolution) of the system output but the advantage is that sampling and frame preparation only use 2% of the frame budget. The only pre-processing that is required when configuring the camera in low-quality mode is converting the image to greyscale. This is done by using the function `cvCvtColor` from the popular computer vision library "OpenCV 2.0"[1].

5.5.2 Interest Point Detection

As discussed in section 4.3, the FAST corner detector provides significant performance increases over other detectors such as Harris and Shi-Tomasi. This performance increase is reflected in our tests comparing FAST to Harris and Shi-Tomasi in section 6.3.1. FAST is preferred by AR researchers such as Klein [18] and Wagner [38]. In section 6.3 our own tests show that FAST performs significantly better than the OpenCV function `cvGoodFeaturesToTrack`. We have ported FAST code to the iPhone 3Gs with great success, detecting hundreds of points in under 10 milliseconds per frame as shown in the tests in section 6.3.2.

While *detecting* hundreds of interest points in real-time is feasible, *tracking* that amount on mobile hardware becomes very challenging. Luckily FAST supports non-maximal suppression and thresholding to limit the amount of points returned as described in section 4.3.2.

[1] `http://opencv.willowgarage.com/wiki/`

We used a FAST threshold of 80 in order to limit the amount of points detected to a more manageable number, in the range of 20 to 40 points per frame. Tracking fewer points leads to reduced robustness and accuracy of pose estimation. We tried to find a threshold that benefits processing time without reducing accuracy too much. In section 6.5.2 we show our test results comparing number of points detected to system speed and tracking accuracy and show that approximately 30 points distributed over an image delivers sufficient tracking accuracy so that a believable AR environment can be rendered. Figure 4.5 shows FAST corner detection applied to an image.

5.5.3 Interest Point Tracking

Our tracking strategy was initially based on Speeded Up Robust Features (SURF) descriptors. SURF is a means of interest point *identification* and is used to match detected points between frames. It is discussed in detail in section 4.4. Even though SURF descriptors worked reasonably well in terms of robustness and accuracy, we soon found that SURF was too computationally intensive and our system was only capable of achieving frame rates of 1-5 frames per second. Our experimental measurements are shown in sections 6.4.1 and 6.4.2. An alternate strategy to interest point description is optical flow tracking, specifically **sparse optical flow** tracking. For our application we used the Lucas-Kanade image-pyramid based sparse optical flow tracker as implemented by `cvCalcOpticalFlowPyrLK` in OpenCV. The tracker and optical flow are discussed in more detail in section 4.5. Figure 4.8 shows the result of `cvCalcOpticalFlowPyrLK` applied to two corresponding images. The results of a number of tests performed on the iPhone 3Gs to determine the speed and consistency of sparse optical flow tracking are shown in sections 6.4.3 and 6.4.4.

We assume steady camera movement which will result in gradual changes across video frames and also assume that consecutive video frames will not differ considerably. Erratic movement - resulting in blurred frames or sudden large changes between frames - will cause tracking to fail. In order to recover from tracking failure, *key-frames* may be carefully chosen to improve the chances of successful re-localisation. A key-frame is an image captured and stored that may be used as a reference to suppress the cumulative error effect of noisy camera sensors over time. A key-frame could also be used to recover from total tracking failure. We were not able to implement a key-frame based solution in time and leave it as an improvement for further work. Adding a key-frame recovery system to our application will benefit overall robustness because it will allow the system to correct its incremental estimate periodically based on key-frame information.

5.5.4 Partial Pose Estimation

As discussed in section 3.7 we only need to use visual tracking to estimate the translation vector, $\vec{t}$ between consecutive frames. Our current strategy for finding $\vec{t}$ uses a sequential frame tracking approach. The pose estimate buffer mentioned in section 5.4 stores three

values related to translation that are the total translations along each axis of the camera relative to an initial starting position:

- Total translation along the camera's x-axis: $t_{x,total}$.
- Total translation along the camera's y-axis: $t_{y,total}$.
- Scale. Scale is equivalent to translation along the camera's z-axis (equation (3.2)): $t_{z,total}$.

Overall translation, $\vec{t}_{Total}$, is then:

$$\vec{t}_{Total} = \begin{bmatrix} T_{x,total} & T_{y,total} & T_{z,total} \end{bmatrix}^T$$

Every time a new frame is processed and a relative change in translation from the previous frame is calculated, $\vec{t}$, we update $\vec{t}_{Total}$ by adding $\vec{t}$:

$$\vec{t}_{Total,i} = \vec{t}_{Total,i-1} + \vec{t}_i$$

Where $\vec{t}_{Total,i}$ is the ith estimate of $\vec{t}_{Total}$, $\vec{t}_{Total,i-1}$ is the previous estimate of $\vec{t}_{Total}$ and $\vec{t}_i$ is the relative change in translation between the current frame and the previous frame.

Calculating Relative Translation Between Frames

In this section we will describe how we calculate the relative change in translation between the current frame and the previous frame relative to the three axes of the camera. $\vec{t}$ is defined as:

$$\vec{t} = \begin{bmatrix} \Delta x & \Delta y & \Delta z \end{bmatrix}^T$$

Where Δx is the relative change in translation between two points, in this case frames, along the camera's x-axis in image coordinates.

When tracking points between consecutive frames using feature descriptors or optical flow, we obtain an array representing the relative movement of points between frames as visualised in figure 4.8. The array is given as a set of (dx,dy) movement values, one for each tracked point. To find Δx and Δy we average all the dx and dy values for two consecutive frames. For n points tracked:

$$(\Delta x, \Delta y) = \frac{1}{n} \sum_{k=1}^{n} (dx_k, dy_k)$$

This method is crude and easily influenced by outliers and noise, however it is very fast and works well for mostly 2-D translations as shown in our system accuracy tests in section 6.5.2. In chapter 7 we list outlier detection as possible future work.

Finding the change in scale between two consecutive frames (Δz) is slightly trickier. We found that when the camera is translated along its z-axis that the movement of points in the

top and left segments of an image move in the opposite direction relative to points in the bottom and right segments. We split the array of (dx, dy) values up into four quadrants:

$$Q1 = (dx_1[1], dy_1[1]), (dx_1[2], dy_1[2]), ...$$

$$Q2 = (dx_2[1], dy_2[1]), (dx_2[2], dy_2[2]), ...$$

$$Q3 = (dx_3[1], dy_3[1]), (dx_3[2], dy_3[2]), ...$$

$$Q4 = (dx_4[1], dy_4[1]), (dx_4[2], dy_4[2]), ...$$

With $Q1$ being the top left quadrant of the image, $Q2$ top right, $Q3$ bottom left and $Q4$ bottom right. We then find Δx and Δy per quadrant:

$$(\Delta x_1, \Delta y_1) = \langle Q1 \rangle$$

$$(\Delta x_2, \Delta y_2) = \langle Q2 \rangle$$

$$(\Delta x_3, \Delta y_3) = \langle Q3 \rangle$$

$$(\Delta x_4, \Delta y_4) = \langle Q4 \rangle$$

where the angle brackets represent averaging over the arrays $Q1$, $Q2$, $Q3$ and $Q4$.

We then find Δz as follows:

$$\Delta z = (\Delta x_1 + \Delta x_4) - (\Delta x_2 + \Delta x_3)$$

We do not use the Δy values in this calculation.

Again this is a very crude method of calculating the relative difference in scale between frames, but just like the calculation of Δx and Δy it works well for mostly 2-D translations and executes in a negligible amount of time. The accuracy of this method of calculating scale is tested in section 6.5.2. In section 3.5 where we discuss full pose estimation we indicate that full pose requires that at least eight points are tracked. Strictly speaking, our method can estimate translation and scale by tracking only two good points of interest, one in the left half of the image and one in the right. The more points that are accurately tracked, the better our average translation and scaling estimate will be.

5.6 Accelerometer Aided Orientation

Estimating $\mathbf{R}$ and completing the pose estimate using an integrated accelerometer is discussed in detail in section 3.6.1. We configured the accelerometer to return the acceleration vector, $\vec{a}$ at a rate of 100 Hz, or once every 10 milliseconds. The AR scene's orientation is updated every new frame that is rendered (between 30 and 60 frames per second).

5.7 OpenGLES Output Subsystem

The iPhone OpenGLES environment is very similar to the traditional OpenGL implementation prevalent on computer systems. During initialisation an OpenGLES "view" is created and set up by configuring lighting and setting up the OpenGL projection matrix to correspond with screen dimensions.

During normal operation, very little actual work is done by die OpenGLES renderer:

1. The data in the **pose estimation buffer** are retrieved and stored in local variables.
2. The model view matrix is *translated* and *scaled* according to the visual tracking results obtained in section 5.5.4 and contained in the pose estimate buffer.
3. The matrix discussed in section 3.6.1, is passed to the OpenGL function `glMultMatrixf((GLfloat*) R );` to transform the model view matrix and reflect device orientation.
4. A series of filled triangles are drawn according to the vertices of a model as specified in a separate header file (a teapot in the case of our demo application) using `glDrawElements`.
5. Optionally, a grid is drawn using an array of grid vertices passed to `glDrawArrays`.

Because accelerometer updates occur far more frequently than translation and scaling updates, the rendered graphics will reflect changes in rotation at high frame-rates (30-60 frames per second) while translation and scale updates occur at approximately 10 frames per second.

The rendered graphics are drawn on the OpenGLES "view" with transparent background. This view is overlaid on top of a camera preview view to complete the effect of Augmented Reality.

5.8 Summary

In this chapter we discussed our implementation of computer vision techniques on a mobile device in order to estimate $\vec{t}$, as well as making use of an integrated accelerometer sensor to estimate $\mathbf{R}$. Pose is estimated by combining this information within the OpenGL context. In the following chapter we will present our tests and results for specific computer vision techniques and overall system.

Chapter 6

Testing and Evaluation

6.1 Introduction

As part of our objectives for this thesis, specified in 1.5, we evaluated a number of computer vision methods on our chosen mobile platform. The order of testing follows the order of introduction in chapter 4. We evaluate the sample rate of an iPhone 3Gs, the time it takes to prepare a frame for further processing, interest point detection, descriptor calculation and matching, and optical flow tracking. Finally we evaluate the time it takes to estimate pose and accuracy of the implemented prototype system. All tests are performed on an iPhone 3Gs.

6.2 Frame Acquisition and Preparation

In this section we will examine the time it takes to sample and prepare video frames to be used by computer vision tracking algorithms such as interest point detection, description and optical flow tracking.

6.2.1 Device Sample Rate

Motivation

As discussed in section 5.3 we chose an iPhone 3Gs as development platform. The 3Gs can sample incoming video frames from its camera at three different resolutions and frame-rates. The camera can be configured in one of three *camera modes*, **high quality**, **medium quality** and **low quality**. We need to determine which camera setting will work the best for our AR application. In this test we determine the resolution and frame-rate of each camera setting. The frame interval is the inverse of the frame-rate (the time between incoming frames). We also need to determine the sample time which is how long it takes to copy the pixels of an incoming frame into a buffer for further processing. In section 5.2 we defined our "frame budget" as 100 milliseconds. Sample time must be kept to a minimum and will

	High Quality		**Medium Quality**		**Low Quality**	
Resolution	640x480		480x360		192x144	
	focused	focusing	focused	focusing	focused	focusing
Frame-rate	25 fps	15 fps	29 fps	14 fps	15 fps	7 fps
Frame interval	39.8 ms	65.6 ms	34.8 ms	69.9 ms	66.6 ms	143 ms
Sample time	11 ms	7.3 ms	5.5 ms	3.3 ms	0.6 ms	0.5 ms

Table 6.1: *Comparison of Frame-rates, frame intervals and sample times for three different camera modes.*

have the biggest effect on our choice of camera mode. Camera quality and resolution is of secondary importance.

Test Configuration

In order to determine the average frame-rates, frame intervals and sample times for each camera mode we wrote an application for the 3Gs that measures the time between frames and frame sample times of a camera video feed in milliseconds. We found that frame-rate is influenced by the state of the camera. While focusing, the camera would return approximately half the frame-rate returned compared to the frame-rate when focused. For each of the camera modes we took ten samples while recording video, five of a scene in focus and five of a scene out of focus. Out of focus scenes were acquired by covering the device's camera with a cloth so that its auto-focus mechanism was unable to function. The raw sample data is available in Appendix A.

Results

Table 6.1 shows the average frame-rates, frame intervals and frame sample times for the three capture settings.

Discussion

We see that the frame-rate of the low-quality camera setting is approximately half that of the medium- and high-quality settings. This is expected since low frame-rates is a characteristic of low-quality video settings. We see that the sample time of frames returned by a high quality-video feed takes nearly a third of the frame interval for the high-quality setting, although it is only 11% of our frame-budget as discussed in section 5.2. We would like to minimise sample-time to make the maximum amount of processing time available to other computer vision functions. We will choose the low-quality camera setting for our application. Even though the frame-rate of the low-quality setting is lower than the other two settings,

it is still higher than our target frame-rate of 10 frames per second. Choosing low resolution input frames may also remove the need for down-sampling as discussed in section 4.2. This is investigated in the following test.

6.2.2 Frame Preparation

Motivation

As discussed in section 4.2 we need to prepare incoming video frames before passing them to computer vision methods. One of the most important preparations is converting incoming video frames to greyscale, also known as calculating image intensity. Furthermore, we may choose to downscale the image to reduce the computational requirements in further computer vision processing. In this test we will determine the total processing time of sampling a video frame, converting it to greyscale and down-sampling the greyscale version for three camera modes in order to determine which mode would use up the least amount of frame-budget (section 5.2).

Test Configuration

In order to determine the amount of time required to sample, convert and down-sample an incoming video frame we wrote an application that measures the amount of time each respective step takes for incoming video frames. We used a live video feed, not prepared images or a prepared video sequence. For each of the camera modes we took five samples of a scene in focus. We are not interested in the processing time of unfocussed frames. The raw sample data are available in Appendix A.

Results

Figure 6.1 shows the total amount of time required to prepare a single incoming video frame.

Discussion

From the figure we can see that the total amount of time required to prepare an image of the lowest resolution is negligible (less than 2 ms) compared to frames of higher resolutions. Sampling a frame of the highest resolution, converting it to greyscale and down-sampling it only once already takes up nearly 80 ms, or 80% of the 100 ms frame budget discussed in section 5.2. This is the second indication that working with the highest quality incoming video frames should be avoided and that we should work with the low-quality camera mode instead.

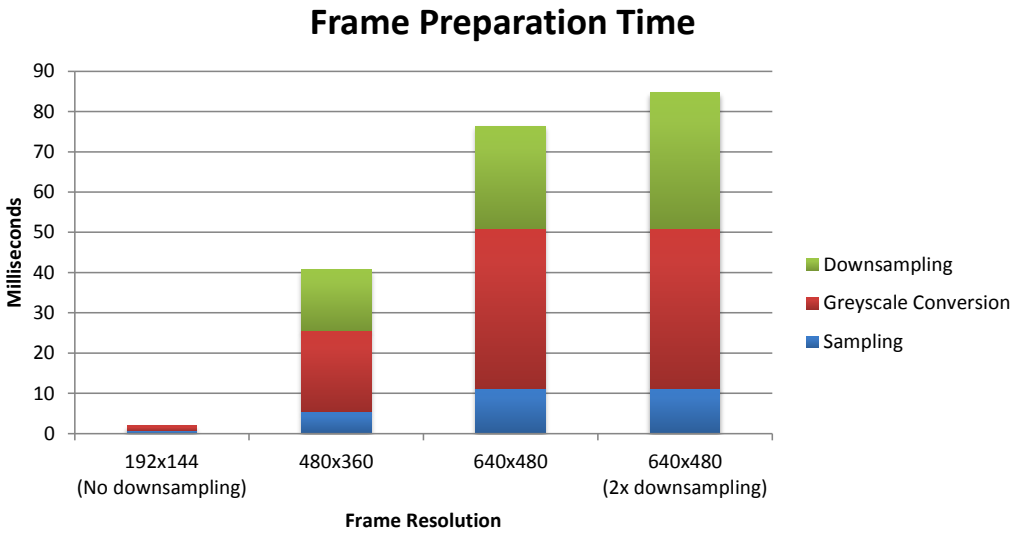


Figure 6.1: *Frame preparation times at various input resolutions.*

6.3 Interest Point Detection

In this section we will compare the speed of four interest point detection functions: Harris [11], Shi-Tomasi [35], FAST [32] without non-maximal suppression and FAST with non-maximal suppression. We will also examine the speed and consistency of the FAST corner detector with non-maximal suppression for various camera modes.

6.3.1 Comparison of Interest Point Detectors

Motivation

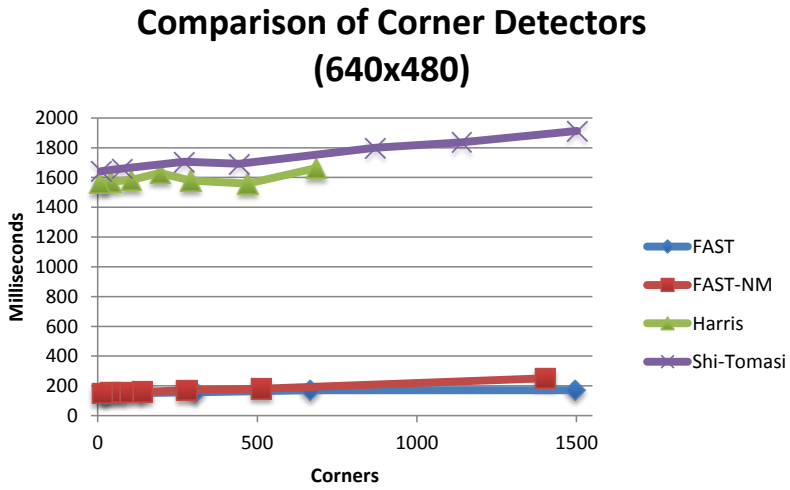
As discussed in section 3.7 we need to find a suitable interest point detector for visual tracking. We will compare the speed of four main interest point detectors discussed in section 4.3.2 namely Harris, Shi-Tomasi, FAST and FAST with non-maximal suppression. In this test we need to find a corner detector that can process an incoming video frame within our frame-budget of 100 ms and still leave a portion of the frame-budget for further computer vision functions.

Test Configuration

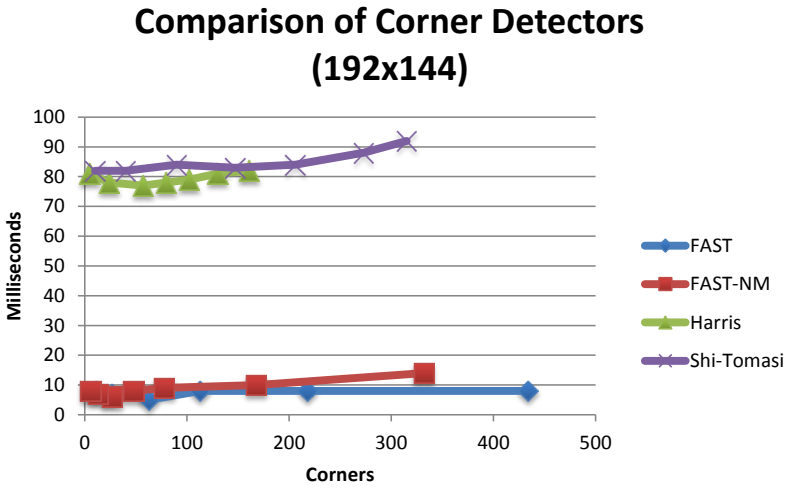
We wrote an application that determines the processing time required for four interest point detection methods respectively. Two images of the same scene were obtained from the iPhone 3Gs's camera at the highest- and lowest camera quality modes and were used for all tests. The images are available in Appendix B as figures 1a and 2a. The detection threshold for each detection method was varied to produce a varying number of corners and measure the processing time for each threshold. The raw measurements are available in Appendix A.

Results

Figure 6.2 shows a comparison of four corner detectors, FAST [32], FAST with Non-maximal suppression, Shi-Tomasi [35] and Harris [11] on two images, one with a resolution of 640 x 480, and another with resolution of 192 x 144.



(a) Comparison of four corner detectors on a 640x480 pixels image.



(b) Comparison of four corner detectors on a 192x144 pixels image.

Figure 6.2: *Device performance characteristics comparison charts.*

Discussion

As seen in the figures, the Shi-Tomasi and Harris detectors are between 8- and 9 times slower than FAST and FAST with non-maximal suppression. Harris may be slightly faster than Shi-Tomasi because it avoids explicit calculation of eigenvalues as discussed in section 4.3.2, however for a given detection threshold it produced nearly half the number of corners of

Shi-Tomasi. This is expected as Shi-Tomasi is an adapted form of Harris but with a "better" corner criteria, as discussed in section 4.3.2. The difference between FAST and FAST with non-maximal suppression is discussed in section 4.3.2. In that section we predicted that non-maximal suppression may incur a small performance penalty. This is evident in figure 6.2 where the calculation time for FAST with non-maximal suppression increases with the number of corners detected. From figure 6.2 we see that Harris and Shi-Tomasi takes up too much of our 100 ms frame-budget, even for the lowest quality camera mode. In section 6.2.2 we see that FAST with non-maximal suppression is only a couple of milliseconds slower than FAST, even for more than 300 corners detected. The advantage of choosing FAST with non-maximal suppression over normal FAST is discussed in section 4.3.2. Given the speed of FAST and the advantages of non-maximal suppression, we will use FAST with non-maximal suppression as our interest point detector in our prototype AR application.

6.3.2 FAST at Different Resolutions

Motivation

In our previous test in section 6.3 we show that FAST with non-maximal suppression is suitable for our application, and the most feasible of the four methods compared. In this section we will examine the performance of FAST with non-maximal suppression in more detail in terms of speed on a number of frame resolutions. The results of this test will not influence our choice of interest point detector, we only include it for comparison.

Test Configuration

We produced a number of images at varying resolutions based on figure 1a in Appendix B with a resolution of 640x480 and figure 2a with a resolution of 192x144. The following resulting resolutions were obtained:

- **640x480**: The original image of figure 1a was used.
- **480x360**: The same scene as in figure 1a was sampled using the camera's "medium-quality" mode.
- **240x180**: The image of resolution 480x360 was down-sampled once.
- **192x144**: The original image of figure 2a was used.
- **160x120**: The image of resolution 640x480 was down-sampled twice.

On each of these images we applied a FAST corner detector with non-maximal suppression with varying detection threshold in order to vary the number of corners detected and measured the calculation time of each step for each image. The raw measurements are available in Appendix A.

Results

Figure 6.3 shows the result of the measurements.

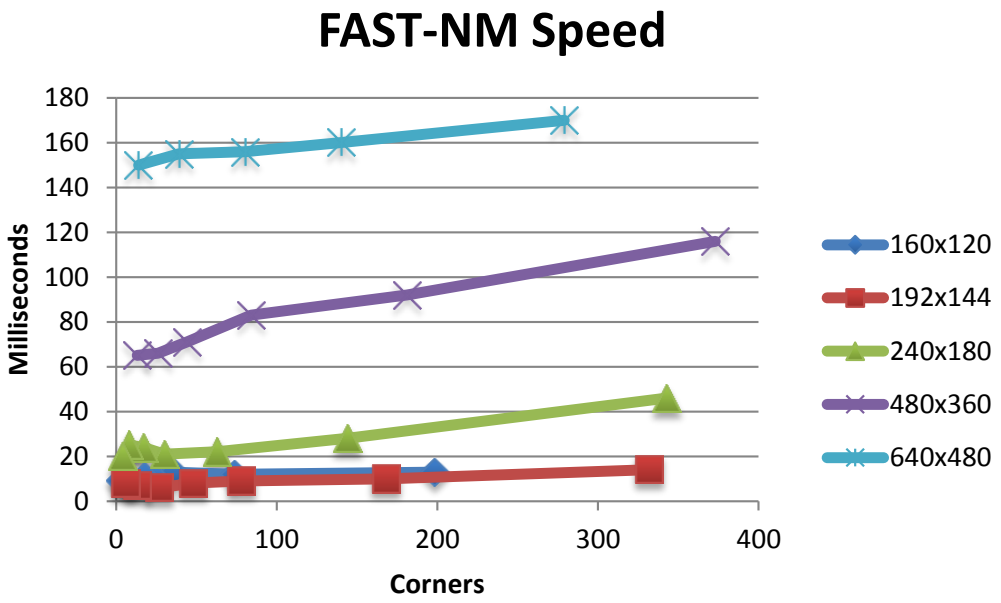


Figure 6.3: *FAST with non-maximal suppression detection speed.*

Discussion

Due to the non-maximal suppression step, the detection time increases slightly with an increasing number of corners detected by the FAST detection step because a score must be calculated for each corner candidate. From the graphs we see that FAST detection on an image of highest resolution takes longer than the available frame-budget of 100 ms discussed in section 5.2. The medium quality camera mode also takes too long, but down-sampled once the calculation time becomes acceptable. Frames of the lowest quality camera mode or of the highest quality camera mode down-sampled twice produces results that are well within our frame-budget. Detecting between zero and 300 corners in a frame of resolution 192x144 takes less than 20 ms.

6.3.3 FAST Consistency

Motivation

A corner detector would not be very effective if it simply chose pixels as corners at random. One of the most important aspects of a corner detector is that it must detect the same points over different views of the same scene. Another way to describe this is that a corner detector must be *consistent*. We need to determine whether the FAST corner detector with non-maximal suppression is consistent enough for use in our tracker.

Test Configuration

We produced five images of the same scene but from different orientations for two camera quality modes, high-quality and low-quality. The high-quality images are available in Appendix B as figure 1 and the low-quality images as figure 2. The second image of each set was scaled relative to the first by a factor of approximately 1.5. The third image was translated relative to the first by 7 cm. The fourth image was rotated approximately 15 degrees clockwise relative to the first. The fifth image was rotated approximately 50 degrees counter-clockwise relative to the first.

We used the FAST detector with non-maximal suppression to detect corners in each image and verified corresponding corners between the original and changed images by hand. The test was performed on images of the highest quality and lowest quality camera modes. The raw measurements are available in Appendix A.

Results

Figure 6.4 shows the result of the measurements.

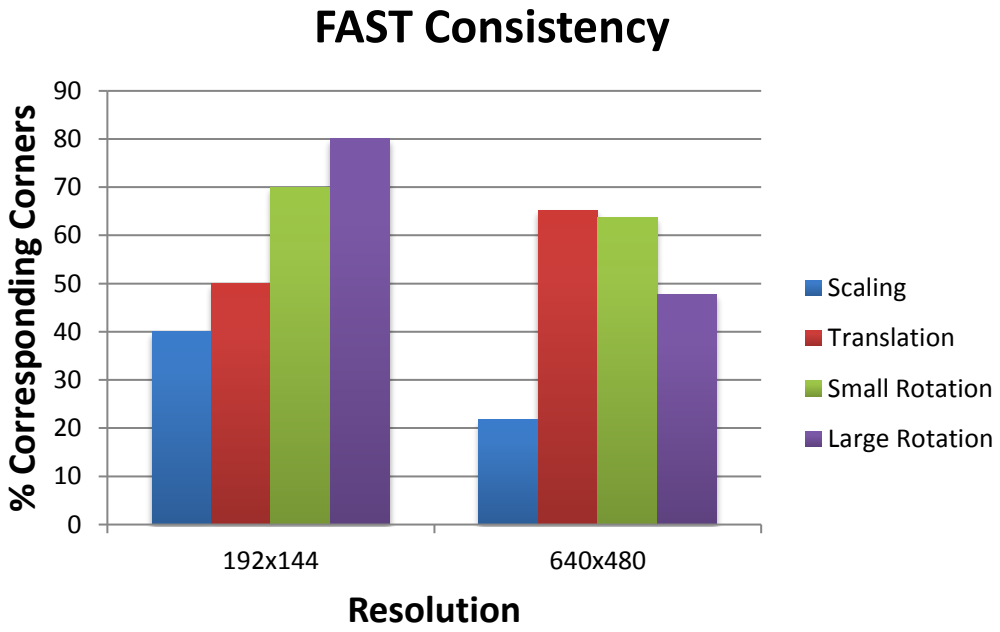


Figure 6.4: *FAST with non-maximal suppression detection consistency.*

Discussion

From the figure we see that we can expect FAST to detect corresponding corners in at least 50% of cases for translations, small rotations and large rotations. FAST is not very robust against changes in scale. As discussed in section 5.5.3 we will track points from frame to frame and assume small changes between frames. This test shows that FAST should be able

to find consistent points between frames for small changes and is therefore suitable for use in our application.

6.4 Interest Point Tracking

In this section we evaluate SURF [4], Upright-SURF(USURF) and Lucas-Kanade [25] sparse optical flow tracking as potential strategies for tracking points of interest as detected by a FAST corner detector with non-maximal suppression. We need to find a tracking strategy that will not exceed our frame budget of 100 ms (section 5.2) between frames. From section 3.5 we know that we need to track at least eight points between frames for accurate computer vision based pose estimation. For our partial pose estimate discussed in section 5.5.4 we only need to track at least two points, one in the left half of the image and one in the right. The more points that are tracked, the better our pose estimate will be. We impose a minimum limit of eight points to track so that we may later substitute our own partial pose estimation method with a more advanced method that might require the minimum number of points for full pose estimation. Bay [4] already performed tests to show the increase in performance of SURF over SIFT [24] so we will not perform any tests related to SIFT.

6.4.1 Descriptor Calculation and Matching Speed

Motivation

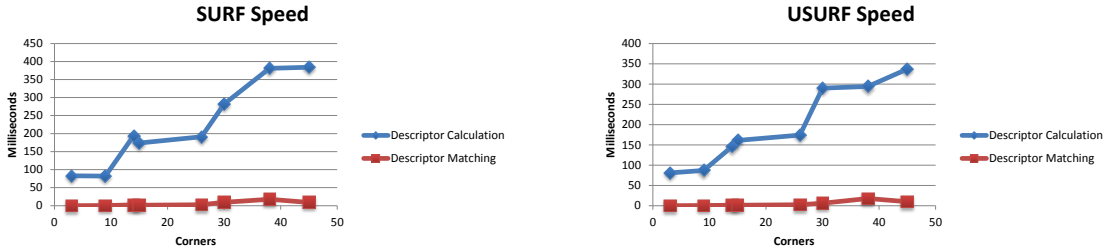
SURF [38] needs to calculate *descriptors* that uniquely describe corners detected by a corner detector in order to match corners between frames. SURF is discussed in section 4.4. Upright-SURF (USURF) is SURF but with the step determining the orientation of a descriptor removed. Removing this step means that USURF is slightly faster than normal SURF, but it is not rotation invariant and requires that an interest point not be rotated more than $\pm$ 15 degrees [4]. We will perform a test determining the calculation and matching time required by SURF and USURF for tracking.

Test Configuration

For this test we used the low-quality (192x144) images shown in Appendix B, figure 2. We used a FAST corner detector with non-maximal suppression and varied the detection threshold to detect a varying number of points. For each threshold, we matched the corners detected in the figure 2a to the corners detected in the other four images using SURF and USURF. We only measured the feature descriptor and matching times, leaving out frame preparation and corner detection times. This produced a table of results relating SURF and USURF descriptor calculation and matching time to the number of corners detected. The table is available in Appendix A.

Results

Figure 6.5 shows the result of the measurements.



(a) SURF descriptor calculation and matching speed.

(b) Upright-SURF descriptor calculation and matching speed.

Figure 6.5: *SURF and USURF descriptor calculation and matching speed.*

Discussion

From figure 6.5 we can see that calculating as few as 20 SURF or USURF descriptors already consume almost twice our frame-budget of 100 ms. Descriptor matching is almost instantaneous taking less than 1 ms in most cases. These results indicate that we will not be able to use SURF as-is for tracking interest points in our AR application. Comparing figure 6.5a and figure 6.5b we see that USURF does not perform much better than SURF and is therefore not suitable for our application either.

6.4.2 Descriptor Consistency

Motivation

To be able to compare SURF and optical flow we measure the robustness of SURF against various transformations such as scaling, translation and rotation.

Test Configuration

We used the same test images generated for section 6.3.3 (scaled, translated, small rotation, large rotation). For each transformation we measured the amount of corners correctly matched by SURF. We performed the test at two resolutions, the lowest- and highest quality camera modes. The raw measurements are available in Appendix A and the test images in Appendix B.

Results

Figure 6.6 shows the result of the measurements.

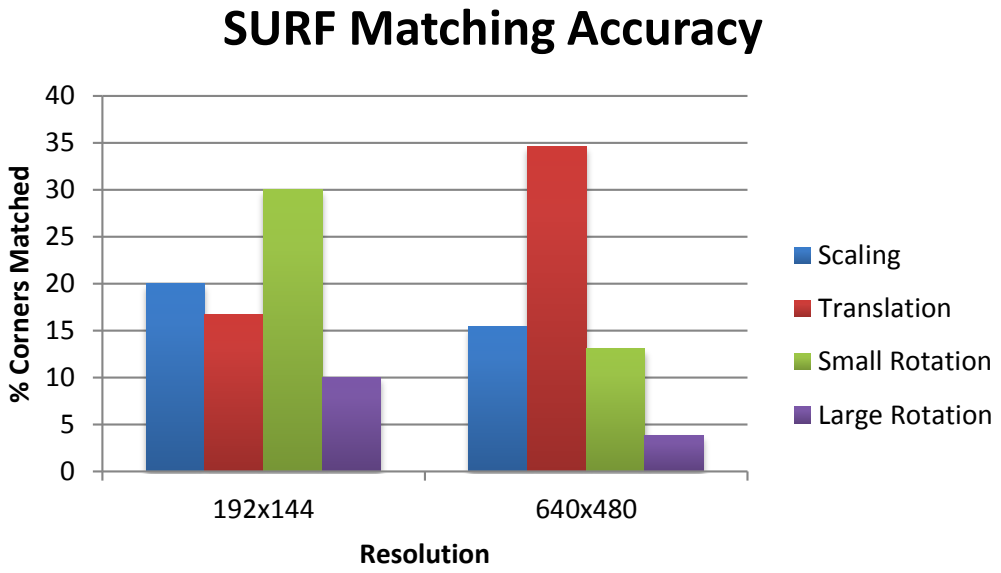


Figure 6.6: *SURF descriptor calculation and matching accuracy.*

Discussion

From the figure we see that the amount of corners accurately matched between frames are between 10% and 35%. This percentage is heavily affected by the consistency of FAST as discussed in section 6.3.3. At the start of section 6.4 we indicate that we need to track at least eight points. From figure 6.6 we see that this means we would need to calculate descriptors for between 20 and 80 points each frame. As shown in section 6.4.1, that is far too computationally expensive for an iPhone 3Gs to handle.

6.4.3 Optical Flow Speed

Motivation

As discussed in section 4.5, sparse optical flow tracking may be used to track the motion of points between two frames instead of calculating feature descriptors for both frames and matching the descriptors. There are also some optimisations that may allow optical flow tracking to perform significantly faster than SURF if used correctly. These optimisations are discussed in the "discussion" section below the test results. We need to determine the difference in performance between optical flow tracking and descriptor based point tracking.

Test Configuration

In order to test the speed of the Lukas-Kanade sparse optical flow tracker, we used the same test configuration as for section 6.4.1. Where SURF first calculates descriptors for one frame and then for the next and then match the descriptors later, optical flow takes two images

as input and immediately finds point correspondences between the images. We supplied the OpenCV function, `cvCalcOpticalFlowPyrLK` with the same two images and corners as in section 6.4.1 and measured the time it took to complete given a varying number of FAST corners. The raw measurements are available in Appendix A.

Results

Figure 6.7 shows the result of the measurements. "Corners" is the total number of corners detected in the last frame.

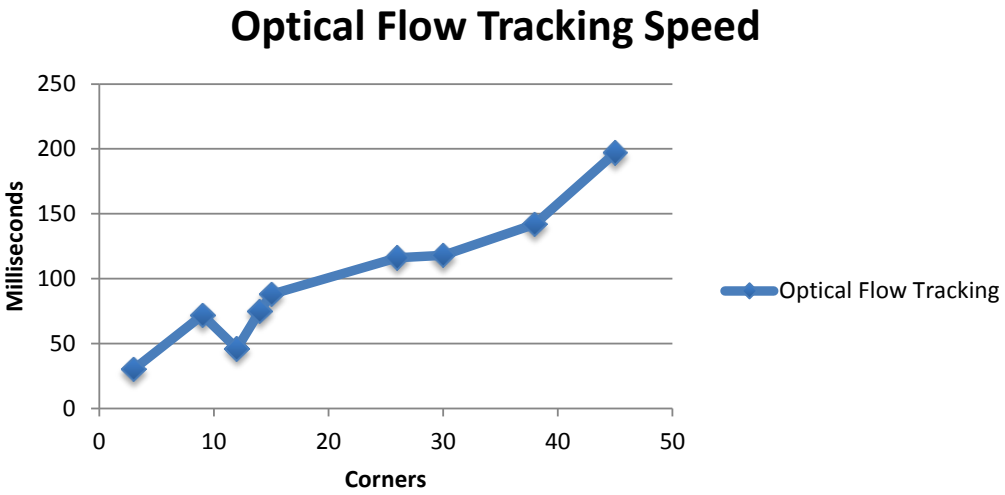


Figure 6.7: *Sparse optical flow tracking speed.*

Discussion

The graph shows that tracking sparse optical flow is only possible in real-time using fewer than 20 corners. There are, however, two optimisations that can improve the performance of `cvCalcOpticalFlowPyrLK`. Information from a previous calculation of sparse optical flow using this function can be used in the following calculation thereby reducing overall processing time. Additionally a model of the system, such as an Extended Kalman Filter (EKF) as discussed in section 4.5, may be used to "predict" the approximate new positions of points. Using an EKF to predict pose falls outside the scope of this thesis, but its use is listed as possible further work in chapter 7. The downside of Optical flow is that it searches for moved points within a search window. If a point has moved outside the search window it can not be tracked. Comparing figure 6.5 and figure 6.7 we see that optical flow tracking is approximately twice as fast as descriptor based tracking. Combined with the optimisation of reusing information from previous frames it should allow us to operate within our frame-budget of 100 ms.

6.4.4 Optical Flow Consistency

Motivation

As discussed at the start of section 6.4 we need to track at least eight points between frames. We need to test the consistency of optical flow tracking in order to determine whether it will allow us to track at least eight corners within the frame-budget of 100 ms.

Test Configuration

We used the same test configuration as in section 6.4.2 to test the consistency of optical flow. We used the same test images generated for section 6.3.3. One image is scaled by a factor of 1.5, one image is translated by 7 cm, one image is rotated by approximately 15 degrees clockwise and one image is rotated by approximately 50 degrees counter-clockwise. We used the same corners detected in section 6.4.2 and measured the percentage of corners matched between frames. We performed the test at two image resolutions, the highest- and lowest quality camera modes. The raw measurements are available in Appendix A and the test images in Appendix B.

Results

Figure 6.8 shows the result of the measurements.

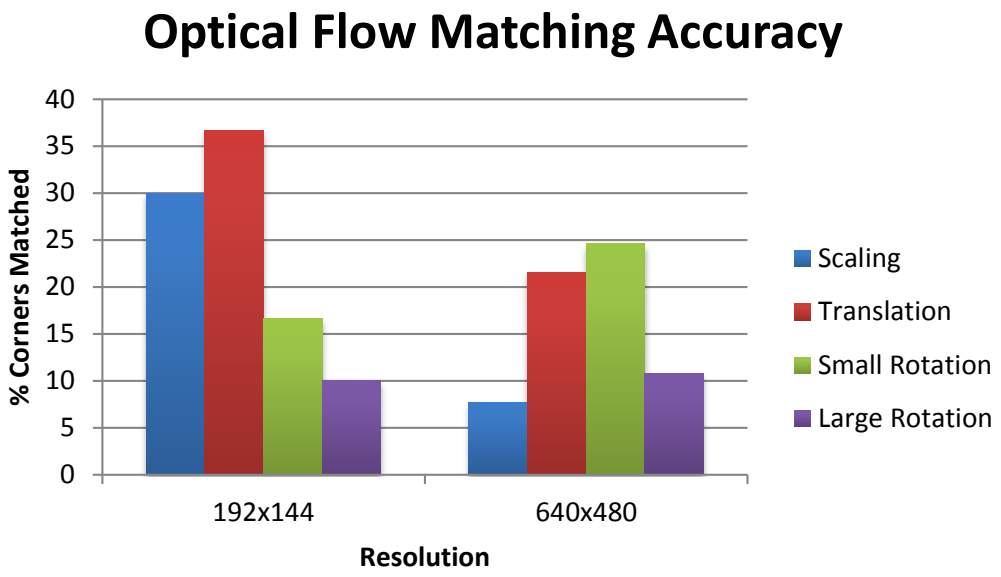


Figure 6.8: *Sparse optical flow tracking consistency.*

Discussion

From the graph we see that optical flow does not perform well for large rotations. This is expected since one of the assumptions the Lucas-Kanade tracker makes is that points only travel small distances between frames as discussed in section 4.5. We see that the tracking consistency for the higher resolution images is worse than for the lower resolution images. This could be due to there being a higher chance of corners falling outside the optical flow tracker's search window in a higher resolution image. On the lower resolution images, the optical flow tracker compares favourably to a SURF based tracking strategy in terms of consistency. With a consistency of between 15 % and 35 % for changes in scale and translation we need to detect at least 20 to 50 points per frame.

6.5 Pose Estimation

6.5.1 Pose Estimation Speed

Motivation

Pose estimation includes frame preparation, corner detection, corner tracking, estimating $\vec{t}$ and combining $\vec{t}$ and $\mathbf{R}$ as described in section 4.6. We need to measure the time it takes to complete all these steps in order to determine whether our application can operate within the 100 ms frame-budget and in real-time.

Test Configuration

We did not measure the time it takes to estimate $\vec{t}$ and combine it with $\mathbf{R}$ because the processing time is negligible (under 1 ms), requiring only a few addition and multiplication operations. For this test we implemented all the steps necessary for partial pose estimation as described in section 3.7:

- Frame Sampling: We used the low-quality camera mode returning video frames at a resolution of 192x144 pixels at 15 frames per second.
- Frame Preparation: We only convert the sampled image to greyscale. We do not do any down-sampling.
- Interest Point Detection: We use the FAST corner detector with non-maximal suppression with a varying threshold.
- Interest Point Tracking: We use the OpenCV implementation of the Lucas-Kanade sparse optical flow tracker, `cvCalcOpticalFlowPyrLK` .

We used a timer to measure the total time it takes to complete all these steps between incoming video frames for a varying number of corners detected. The raw measurements are available in Appendix A.

Results

Figure 6.9 shows the result of the measurements. "Corners" is the total number of corners detected in the last frame.

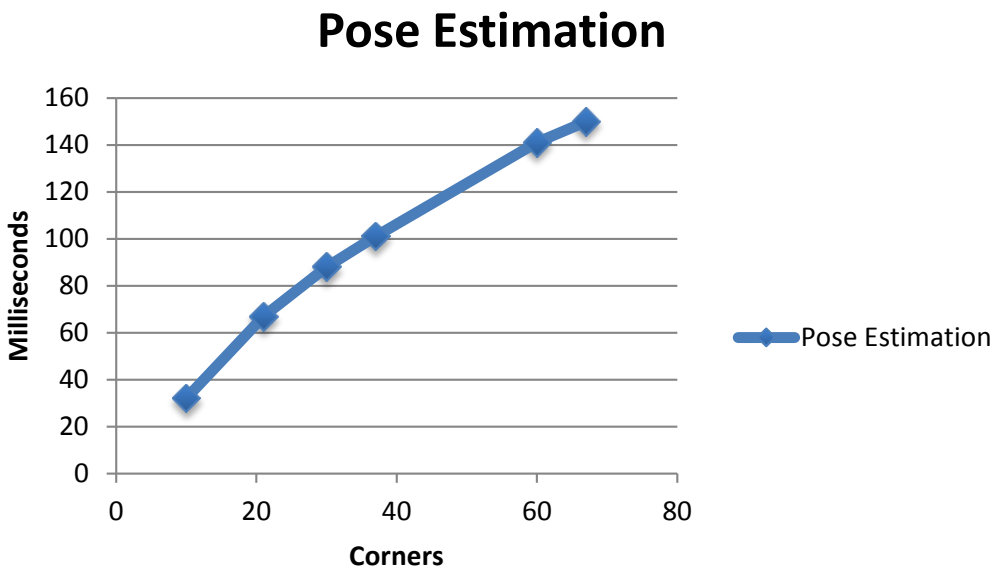


Figure 6.9: *Pose estimation total processing time.*

Discussion

The total processing time for a given number of points is shown in figure 6.9. This time in milliseconds is made up out of the following components:

- Frame Sampling: 0.6 ms (Section 6.2.1).
- Frame Preparation: Approximately 2 ms (Section 6.2.2).
- Interest Point Detection: Approximately 10 ms (Section 6.3.2).
- Optical Flow: The greatest influence on the values shown on the graph.
- Partial Pose Estimation: Negligible (Less than 1 ms).

Since we use the lowest quality camera mode, frame preparation time and FAST corner calculation time is almost negligible. Sparse optical flow tracking takes up the bulk of the frame-budget of 100 ms. Even though it may seem that we are tracking more corners in a specific amount of time than is possible as shown in section 6.4.3, this may be due to the optimisation we use as described in section 4.5. Each frame we store the image pyramid calculated for the current frame and then pass it to the `cvCalcOpticalFlowPyrLK` function the next iteration so that it does not have to be recalculated.

From the graph we see that choosing the threshold for the FAST detector to detect between 20 and 50 corners per frame as specified in the Optical Flow Consistency test (section 6.4.4) allow us to perform pose estimation without exceeding the frame budget by more than 20%.

6.5.2 Pose Estimation Accuracy

Motivation

The accuracy of orientation estimation depends on the accuracy of the accelerometer readings. For a factory calibrated iPhone the accelerator readings are accurate beyond any error that we can measure. We only need to determine the accuracy of our tracking strategy and naive partial pose estimation technique described in section 5.5.4 in order to judge the overall effectiveness of our prototype system.

Test Configuration

We performed three transformation tests in order to determine the accuracy of our tracking system: Translation, scaling and "tilting". Translation is defined as movement along the device's x- or y-axis or both. Scaling is defined as movement along the device's z-axis. "Tilting" is defined as rotation around a real-world object at a constant radius with the device's camera always facing the object.

We mounted the phone above and parallel to the surface of a computer keyboard which provides a good texture to track. We placed a marker of a different colour than the keyboard keys on top of the keyboard and rendered a grid on the view of the camera feed so as to be centred on the coloured marker on system initialisation. A picture of this configuration is shown in figure 6.10. There is a cross-hair drawn on the marker to identify its centre, but it is not visible in the figure. We then translated the phone first along its x-axis and then

Figure 6.10: *Marker used in pose estimation accuracy tests.*

its y-axis and took a set of samples measuring the actual translation of the marker and the

estimated **translation**. The actual translation and translation error samples are available in Appendix A.

We performed the same test for **scale**, but moved the phone along its z-axis instead of its x- and y-axes. We took a number of samples measuring the estimated scale of the marker object and actual scale. The actual scale and scale error samples are available in Appendix A.

Finally we performed the same test for determining the error as a result of "tilt". Tilt is the rotation of the phone around the *marker*'s x-axis at a constant radius. As the accelerometer provided the correct change in orientation for this test, we only measured the error as an offset in the x- and y-axes of the camera image. The tests results are available in Appendix A.

Results

Figure 6.11 shows the results for the three tests. The translation and tilt errors are shown as offsets in pixels. The scaling error is shown as a percentage of actual scale.

Discussion

The output of the display has a resolution of 480 pixels along the x-axis and 320 along the y-axis. From the translation error test in figure 6.11a we see that our estimation of x- and y-translations are fairly accurate. The largest recorded translation error in terms of x-axis translation was 8 pixels (1.7 %) and the largest error in terms of y-axis translation was 15 pixels (4.7 %).

In terms of scaling our system fared a little worse, ranging from a scaling error below 5 % to just over 30 %. The test results in figure 6.11b show that we can grow or shrink a 3-D AR object by an amount closely related to the change in scale between frames. Even though it may not be very accurate, the desired effect is achieved. It is not as easy for a user to detect a slight error in the size of an object than it is to detect an offset of the object from its correct location.

In terms of "tilt", or rotation around an object at a fixed radius, our system becomes very inaccurate. For tilts over 45 degrees the error in terms of an x- and y-offset quickly grows outside reasonable values.

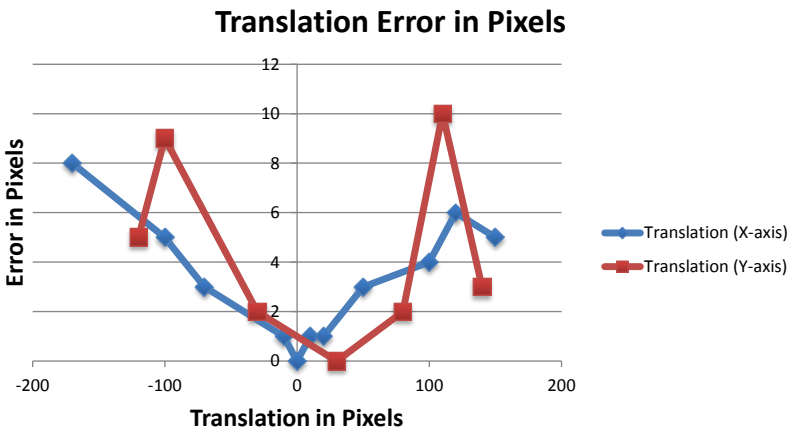
These tests show that even though our system can easily fail under certain circumstances, it performs reasonably accurate translation- and scale-estimations and still functions under small rotations around an object.

6.6 Summary

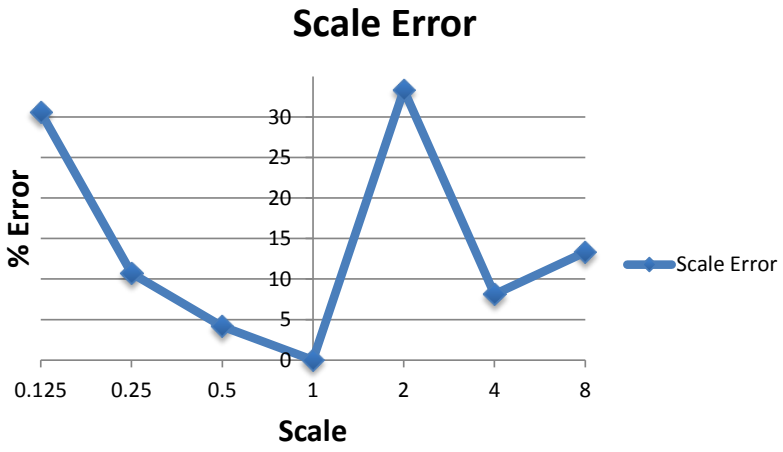
In this chapter we presented a number of tests performed to determine the performance and accuracy of various computer vision techniques. We do not include tests for determining

rendering time and sampling time for the accelerometer. Rendering and accelerometer data sampling does not take up any additional time as it happens on another thread and takes up negligible time (less than 1 ms). A video demonstrating the overall system performance is available at `http://www.youtube.com/watch?v=I4RnDjmGry8`.

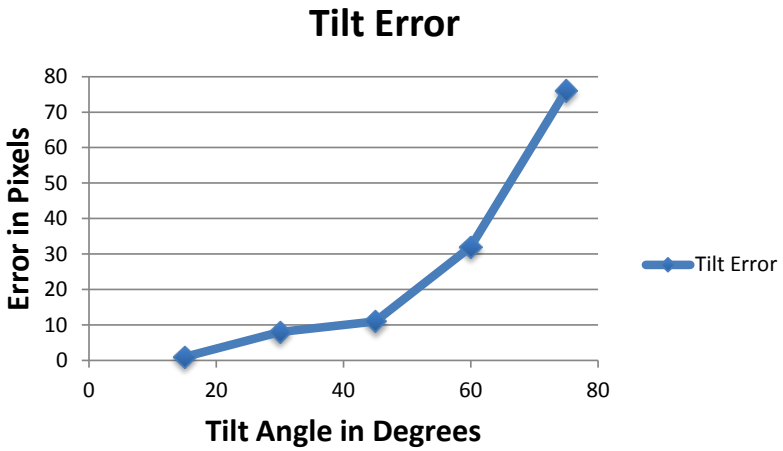
In the following chapter we conclude this thesis, discuss the extent to which we have met our objectives and present possibilities for future work to improve our system.



(a) Translation error



(b) Scale error



(c) Tilt error

Figure 6.11: *Pose estimation accuracy tests.*

Chapter 7

Conclusion

7.1 Introduction

This thesis evaluates the feasibility of developing a hybrid, markerless Augmented Reality application on current-day smart-phone devices. An overview of past research related to computer vision and specifically AR over the past two decades are provided. A number of computer vision techniques have been investigated, implemented and evaluated. The use of integrated sensors to aid orientation estimation was also investigated. In this chapter we revisit our initial objectives to assess whether they have been met and we propose future work that may be pursued to improve upon our results.

7.2 Results

In section 1.5 we identified a set of specific goals for this study:

- **Completion of a theoretical study and review of existing AR research and methods**: In chapters 2 and 4 we present an overview of past work done. Even though this study is by no means complete, we do provide a sample of the complete body of work encompassing AR research. Our overview should give the reader a general understanding of the definition of Augmented Reality, AR platforms and AR techniques.

- **Evaluation of computer vision algorithms on a modern smart-phone platform**: In chapter 6 we present the results of a series of tests related to the various techniques and algorithms involved in computer vision tracking and pose estimation as performed on an iPhone 3Gs. We briefly discuss the results of the tests and discuss how we use the results in the design of our prototype AR application.

- **Fusing integrated sensor data with computer vision to aid pose estimation**: In chapter 5 we discuss a method of transforming AR graphics to reflect the rotation of a device in space according to readings from an attached accelerometer. This method is combined with a naive computer vision tracking strategy to provide full pose tracking.

This strategy relies on a number of assumptions and requires arbitrarily choosing the system's facing direction at initialisation.

- **Implementation of a proof of concept AR application**: Even though we have developed a functioning proof of concept AR application, much work must still be done in order to improve its performance. The system is not very robust and is susceptible to losing track of the AR environment if moved around too quickly. It only barely meets the minimum requirements for real-time performance and its speed may still be improved by optimising the visual tracking strategy.

7.3 Comparison to Prior Work

The only other markerless mobile AR system that we have found for comparison to this thesis is the implementation of "Parallel tracking and mapping on a camera phone" by Klein[19]. Klein implemented a purely visual pose estimation system on an iPhone 3G as part of his post-doctoral research that is far more efficient, robust and advanced than our naive approach to computer vision tracking. However, Klein does not make use of integrated sensors to aid pose estimation and mentions that this could be an avenue of future work in his ISMAR paper[19]. We have implemented such a hybrid system that uses data from an integrated accelerometer to estimate orientation which makes up three of the six degrees of freedom of pose. Our rough computer vision tracking system only estimates relative translation.

Klein uses a keyframe-based SLAM approach as opposed to our iterative frame-to-frame tracking approach. Klein calculates five image pyramid levels for each incoming video frame captured at 320x240 pixels on an iPhone 3G. Klein then performs Shi-Tomasi corner detection on the level two pyramid image of resolution 80x60 and finds the corresponding detected corners on the level one and zero pyramid images. Klein uses bundle adjustments combined with keyframe culling to achieve real-time vision based pose estimation on the iPhone 3G. In order to project points onto the phone's view, Klein uses a homography that is updated from the measurements between frames.

Klein claims a frame-budget of only 60 ms. According to Klein, only 5 ms is required to convert an incoming video frame at 320x240 to greyscale and calculate five image pyramid levels. We have not been able to achieve this level of speed as shown in section 6.2.2, not even on an iPhone 3Gs with a faster processor than the 3G. Klein requires 17.5 ms to search for 50 interest points across three image pyramid levels and only 2.5 ms is used to calculate a pose update. The remaining time per frame is overhead. His results are shown in more detail in[19].

7.4 Further Work

- While this thesis has attempted to exploit the accelerometer now prevalent in many smart-phones, new phones are now emerging that also include digital **gyroscopes**

that may provide much better results. Our accelerometer based orientation estimation method chooses one of the angles of rotation (direction) arbitrarily and this may cause AR models to face the wrong direction. Our method may be adapted to provide all three degrees of freedom of orientation given data from a gyroscope.

- Our system operates at a very low resolution (192 x 144). Apart from improved orientation sensor capabilities, with every new generation of mobile device there is an increase in camera and **video feed quality**. Higher quality video frames combined with greater processing power may offer improved tracking accuracy and robustness in future applications.

- Our visual tracking method is very naive compared to advanced tracking methods such as PTAM presented by Klein [19]. Our sparse optical flow method may be greatly improved with an **Extended Kalman Filter** (EKF) to model camera movement and pose updates according to Bradski [6].

- Currently out system is easily influenced by the presence of noise and sample outliers. Future work may investigate the use of outlier detectors such as RANSAC to reduce these effects.

- Since our system is currently purely iterative, it may be improved with the use of **key-frames** that could allow the system to recover from tracking failure and continuously correct the pose estimate from small, accumulating tracking errors.

7.5 Final Conclusion

In support of our thesis, we present a markerless mobile Augmented Reality prototype application. The application makes use of a hybrid tracking system, fusing computer vision tracking with integrated sensor data to facilitate pose estimation. There are still many improvements and optimisations that may be tested that could delivery smoother frame-rates, more accurate and more robust results. Researching in the visually rich and exciting field of Augmented Reality, and implementing a working prototype, was immensely rewarding.

Bibliography

[1] ADELSON, E. H., ANDERSON, C. H., BERGEN, J. R., BURT, P. J., and OGDEN, J. M., "Pyramid methods in image processing." *RCA engineer*, 1984, Vol. 29, pp. 33–41.

[2] ATCHESON, B., HEIDRICH, W., and IHRKE, I., "An evaluation of optical flow algorithms for background oriented schlieren imaging." *Experiments in fluids*, October 2009, Vol. 46, pp. 467–476.

[3] AZUMA, R. *et al.*, "A survey of augmented reality." *Presence-Teleoperators and Virtual Environments*, 1997, Vol. 6, pp. 355–385.

[4] BAY, H., ESS, A., TUYTELAARS, T., and VANGOOL, L., "Speeded-Up Robust Features (SURF)." *Computer Vision and Image Understanding*, June 2008, Vol. 110, pp. 346–359.

[5] BLESER, G. and HENDEBY, G., "Using optical flow as lightweight SLAM alternative." *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, October 2009, No. 1, pp. 175–176.

[6] BRADSKI, G. and KAEHLER, A., *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[7] CANNY, J., "A Computational Approach to Edge Detection." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986, Vol. 8, No. 6, No. 6, pp. 679–698.

[8] CASTLE, R., GAWLEY, D., KLEIN, G., and MURRAY, D., "Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras." in *Robotics and Automation, 2007 IEEE International Conference on*, vol. 10, pp. 4102–4107, IEEE, 2007.

[9] DAVISON, A. J., "Real-time simultaneous localisation and mapping with a single camera." *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003, Vol. 2, pp. 1403–1410 vol.2.

[10] FISCHLER, M. A. and BOLLES, R. C., "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." *Communications of the ACM*, 1981, Vol. 24, No. 6, No. 6, pp. 381–395.

[11] HARRIS, C. and STEPHENS, M., "A combined corner and edge detector." in *Alvey vision conference* (MATHEWS, M. M. (Ed.)), vol. 15, pp. 147–151, Manchester, UK, Manchester, UK, 1988.

[12] HARRIS, C., *Tracking with Rigid Models*, Ch. 4, pp. 263–284. MIT Press, 1992.

[13] HARTLEY, R. and ZISSERMAN, A., *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[14] HOLMES, S., KLEIN, G., and MURRAY, D., "A square root unscented Kalman filter for visual monoSLAM." in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 3710–3716, IEEE, 2008.

[15] KATO, H. and BILLINGHURST, M., "Marker tracking and HMD calibration for a video-based augmented reality conferencing system." in *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality IWAR99*, vol. 35, pp. 85–94, IEEE Computer Soc. Press, IEEE Comput. Soc, 1999.

[16] KLEIN, G., "Visual tracking for augmented reality." *University of Cambridge, PhD Thesis*, 2006.

[17] KLEIN, G. and MURRAY, D., "Improving the agility of keyframe-based SLAM." *Computer Vision ECCV*, 2008, pp. 802–815.

[18] KLEIN, G. and MURRAY, D., "Parallel tracking and mapping for small AR workspaces." *ISMAR 2007. 6th IEEE and International Symposium on Mixed and Augmented Reality*, 2008.

[19] KLEIN, G. and MURRAY, D., "Parallel tracking and mapping on a camera phone." in *ISMAR 2009. 8th IEEE International Symposium on Mixed and Augmented Reality*, pp. 83–86, IEEE, 2009.

[20] KLEIN, G. and DRUMMOND, T., "Tightly integrated sensor fusion for robust visual tracking." *Image and Vision Computing*, September 2004, Vol. 22, No. 10, pp. 769–776.

[21] KLEIN, G. and DRUMMOND, T., "Tightly integrated sensor fusion for robust visual tracking." *Image and Vision Computing*, 2004, Vol. 22, No. 10, No. 10, pp. 769–776.

[22] KLEIN, G. and DRUMMOND, T., "Tightly integrated sensor fusion for robust visual tracking." *Image and Vision Computing*, 2004, Vol. 22, No. 10, No. 10, pp. 769–776.

[23] LANE, N., MILUZZO, E., LU, H., PEEBLES, D., and CHOUDHURY, T., "A survey of mobile phone sensing." *IEEE Communications Magazine*, 2010, Vol. 48, pp. 140–150.

[24] LOWE, D., "Object recognition from local scale-invariant features." *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, pp. 1150–1157 vol.2.

[25] LUCAS, B., "An iterative image registration technique with an application to stereo vision." *International joint conference on artificial intelligence*, 1981.

[26] MEISNER, J.,
DONNELLY, W., and ROOSEN, R., "Augmented reality technology." September 2003, `http://www.google.com/patents?hl=en&lr=&vid=USPATAPP10664565&id=3jSV`

[27] MELLOR, J. P., "Enhanced Reality Visualization in a Surgical Environment." 1995.

[28] MR3641, "iPhone using the Wikitude application, demonstrating an example of Augmented Reality." 2009,
`http://commons.wikimedia.org/wiki/File:Wikitude.jpg`.

[29] PACKER, R., "Multimedia: From Wagner to Virtual Reality, Expanded Edition." 2002,
`http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/03933`

[30] PERRITAZ, D., SALZMANN, C., GILLET, D., and NAEF, O., "6th Sense Toward a Generic Framework for End-to-End Adaptive Wearable Augmented Reality." *Perception*, 2009, pp. 280–310.

[31] ROSTEN, E., "Machine learning for high-speed corner detection." *Computer Vision, ECCV 2006*, 2006.

[32] ROSTEN, E. and DRUMMOND, T., "Fusing Points and Lines for High Performance Tracking." *Tenth IEEE International Conference on Computer Vision ICCV05*, 2005, Vol. 2, pp. 1508–1515.

[33] ROUSSEEUW, P. J., "Least Median of Squares Regression." *Journal of the American Statistical Association*, 1984, Vol. 79, pp. 871–880.

[34] SHAPIRO, L. and STOCKMAN, G., *Computer Vision*. No. 1. Prentice Hall, 2001.

[35] SHI, J. and TOMASI, C., "Good features to track." in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 94, pp. 593–600, IEEE Comput. Soc. Press, 1994.

[36] TAYLOR, R. H., MITTELSTADT, B. D., PAUL, H. A., HANSON, W., KAZANZIDES, P., ZUHARS, J. F., WILLIAMSON, B., MUSITS, B. L., GLASSMAN, E., and BARGAR, W. L., "An image-directed robotic system for precise orthopaedic surgery." *IEEE Transactions on Robotics and Automation*, 1994, Vol. 10, pp. 261–275.

[37] VERGAUWEN, M. and GOOL, L., "Web-based 3D Reconstruction Service." *Machine Vision and Applications*, May 2006, Vol. 17, No. 6, pp. 411–426.

[38] WAGNER, D., REITMAYR, G., MULLONI, A., DRUMMOND, T., and SCHMALSTIEG, D., "Pose Tracking from Natural Features on Mobile Phones." *ISMAR*, 2008.

[39] WILLIAMS, B., KLEIN, G., and REID, I., "Real-time SLAM relocalisation." *In: Proc 11th IEEE International Conference on Computer Vision (ICCV)*, 2007, Vol. 07, pp. 469–476.

[40] WILLIAMS, B. P. and REID, I. D., "Simultaneous Localisation and Mapping Using a Single Camera." *Philosophy*, 2009.

Appendix A

Sample Intervals (in Milliseconds)

		Sample 1		Sample 2		Sample 3		Sample 4		Sample 5	
Camera Mode	**Focusing**	**Focused**		**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**
low	143.26	66.57		143.13	66.78	143.02	66.53	142.69	66.63	143.28	66.7
medium	66.6	32.59		73.4	35.68	69.23	43.2	73.67	31.28	66.6	31.2
high	66.7	40.2		63.48	36.8	67.5	39.6	66.6	39.4	63.5	43.2

Sample Intervals (in Milliseconds)

		Sample 1		Sample 2		Sample 3		Sample 4		Sample 5	
Camera Mode	**Focusing**	**Focused**		**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**
low	0.55	1.23		0.7	0.55	0.54	0.54	0.71	0.54	0.55	0.55
medium	2.86	6.78		3.04	3.02	5.14	4.53	2.8	7.56	2.76	5.45
high	10.9	13.2		10.5	13.1	5.01	8.48	5.02	11.4	5.03	10.5

Conversion to Greyscale (in Milliseconds)

		Sample 1		Sample 2		Sample 3		Sample 4		Sample 5	
Camera Mode	**Focusing**	**Focused**		**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**
low	1.25	1.24		1.44	1.01	1.19	1.34	1.23	1.24	1.22	1.23
medium	6.5	16		10	14.5	9.8	24	6.5	16	11	25.4
high	22.4	32.4		19.8	37.5	20.4	46.4	19.7	35.5	21.2	46.5

Downsampling (in Milliseconds)

		Sample 1		Sample 2		Sample 3		Sample 4		Sample 5	
Camera Mode	**Focusing**	**Focused**		**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**	**Focusing**	**Focused**
medium	8.5	15.4		7.8	16.5	4.5	21.2	4.8	12.5	4.3	10.6
high (1x)	18	30.1		13.5	25.4	16.2	21.4	10.1	22.5	16.3	27.8
high (2x)	24.5	24.6		19.2	31.5	20.4	36.6	19.4	34.6	21.2	41.2

FAST Speed (in Milliseconds / corners)

Sample:		**FAST1**	**FAST2**	**FAST3**	**FAST4**	**FAST5**	**FAST6**	**FAST7**	**FASTNM1**	**FASTNM2**	**FASTNM3**	**FASTNM4**	**FASTNM5**	**FAST6NM**	**FAST7NM**
	Threshold:	10	30	50	70	90	110	130	10	30	50	70	90	110	130
160x120	**Calculation Time:**	11	14	7	10	9	10	11	13	12	13	10	11	8	9
	Corners Detected:	844	288	132	79	38	20	7	198	74	36	25	18	9	3
192x144	**Calculation Time:**	8	8	8	8	5	7	8	14	10	9	8	6	7	8
	Corners Detected:	1118	434	218	113	63	27	11	332	168	78	48	27	13	6
240x180	**Calculation Time:**	29	30	22	25	25	22	19	46	28	22	21	24	25	20
	Corners Detected:	1472	476	167	80	40	18	7	343	144	63	30	17	8	4
480x360	**Calculation Time:**	135	116	111	105	104	96	65	142	116	92	83	71	66	65
	Corners Detected:	3729	1190	478	230	111	61	23	967	373	181	84	44	26	13
640x480	**Calculation Time:**	181	171	170	157	152	150	145	250	180	170	160	156	155	150
	Corners Detected:	5111	1495	665	307	137	83	23	1402	512	279	140	80	39	14

Interest point detection (640x480)

Sample:	**FAST1**	**FAST2**	**FAST3**	**FAST4**	**FAST5**	**FAST6**	**FAST7**	**FASTNM1**	**FASTNM2**	**FASTNM3**	**FASTNM4**	**FASTNM5**	**FAST6NM**	**FAST7NM**
Threshold:	10	30	50	70	90	110	130	10	30	50	70	90	110	130
Sample Time:	181	171	170	157	152	150	145	250	180	170	160	156	155	150
Corners:	5111	1495	665	307	137	83	23	1402	512	279	140	80	39	14

Sample:	**ST1**	**ST2**	**ST3**	**ST4**	**ST5**	**ST6**	**ST7**	**H1**	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**
Threshold:	0.01	0.02	0.04	0.08	0.16	0.32	0.64	0.01	0.02	0.04	0.08	0.16	0.32	0.64
Sample Time:	1913	1836	1801	1693	1706	1660	1642	1664	1558	1580	1631	1582	1573	1560
Corners:	1502	1140	870	443	272	76	11	684	470	291	195	104	40	6

Interest point detection (192x144)

Sample:	**FAST1**	**FAST2**	**FAST3**	**FAST4**	**FAST5**	**FAST6**	**FAST7**	**FASTNM1**	**FASTNM2**	**FASTNM3**	**FASTNM4**	**FASTNM5**	**FAST6NM**	**FAST7NM**
Threshold:	10	30	50	70	90	110	130	10	30	50	70	90	110	130
Sample Time:	8	8	8	8	5	7	8	14	10	9	8	6	7	8
Corners:	1118	434	218	113	63	27	11	332	168	78	48	27	13	6

Sample:	**ST1**	**ST2**	**ST3**	**ST4**	**ST5**	**ST6**	**ST7**	**H1**	**H2**	**H3**	**H4**	**H5**	**H6**	**H7**
Threshold:	0.01	0.02	0.04	0.08	0.16	0.32	0.64	0.01	0.02	0.04	0.08	0.16	0.32	0.64
Sample Time:	92	88	84	83	84	82	82	82	81	79	78	77	78	81
Corners:	315	273	206	147	90	40	10	161	130	102	80	57	24	5

FAST Consistency:

	Threshold:				**90**					**70**		
		Normal	**Scaled**	**Translated**	**Rotated1**	**Rotated2**	**Normal**	**Scaled**	**Translated**	**Rotated1**	**Rotated2**
Resolution											
192x144	**corners detected**	3	9	3	3	3	10	27	14	12	15
	corresponding corners		3	2	1	2		4	5	7	8
640x480	**corners detected**	19	19	28	16	16	69	46	67	60	52
	corresponding corners		5	14	11	10		15	45	44	33

SURF:

	FAST Threshold:			**50**					**70**			
Resolution		**Scaled**	**Translated**	**Rotated1**	**Heavy Rotation**	**Set**	**Scaled**	**Translated**	**Rotated1**	**Heavy Rotation**	**Set**	
192x144	**time calc**	384	381	382	383		191	194	181	174		
	time match	9.1	17.6	9.3	5.4		2.4	2.2	1	1.2		
	corners detected	45	38	30	26		26	14	12	15		
	matches	6	5	9	3		6	2	4	1		
	correct matches											
640x480	**time calc**	2376	2363	2374	2398	16020	1113	1094	1098	1071	6151	
	time match	225	310	283	382	14494	43	60	58	53	1489	
	corners detected	102	143	131	140	918	46	67	60	52	301	
	matches	2	45	17	0	61	8	20	10	1	16	

	FAST Threshold:		**90**			
Resolution		**Scaled**	**Translated**	**Rotated1**	**Rotated2**	**Set**
192x144	**time calc**	81.98	82	81	83	
	time match	0.4	0.12	0.14	0.1	
	corners detected	9	3	3	3	
	matches	3	2	1	0	
	correct matches					
640x480	**time calc**	400	369	397	398	2609
	time match	3.7	4.9	3.5	2.6	186
	corners detected	19	28	16	16	125
	matches	0	6	5	0	3

USURF:

	FAST Threshold:			**50**				**70**					**90**		
Resolution		**Scaled**	**Translated**	**Rotated1**	**Heavy Rotation**		**Scaled**	**Translated**	**Rotated1**	**Heavy Rotation**		**Scaled**	**Translated**	**Rotated1**	**Rotated2**
192x144	**time calc**	337	294	323	298		174	146	158	161		87	68	76	81
	time match	9.9	16.2	6	9.3		2.23	1.13	1.07	1.2		0.3	0.12	0.11	0.1
	corners detected	45	38	30	26		26	14	12	15		9	3	3	3
	matches	10	7	4	2		7	3	2	2		2	2	1	0.1

Optical Flow:

	FAST Threshold:			**50**					**70**			
Resolution		**Scaled**	**Translated**	**Rotated1**	**Heavy Rotation**	**Set**	**Scaled**	**Translated**	**Rotated1**	**Heavy Rotation**	**Set**	
192x144	**time match**	197	142	118	116		122	75	46	88		
	corners detected	45	38	30	26		26	14	12	15		
	matches	9	11	5	1		5	3	2	1		
640x480	**time match**	801	886	786	918	5861	397	377	375	294	1924	
	corners detected	102	143	131	140	918	46	46	60	52	301	
	matches	10	28	32	14	48	5	5	11	4	13	

	FAST Threshold:			**90**		
Resolution		**Scaled**	**Translated**	**Rotated1**	**Rotated2**	**Set**
192x144	**time match**	72	62	30	21	
	corners detected	9	3	3	3	
	matches	1	0	1	0	

640x480	**time match**	188	210	137	100	814	
	corners detected	19	28	16	16	106	
	matches	3	1	2	3	2	

Pose Estimation Speed:

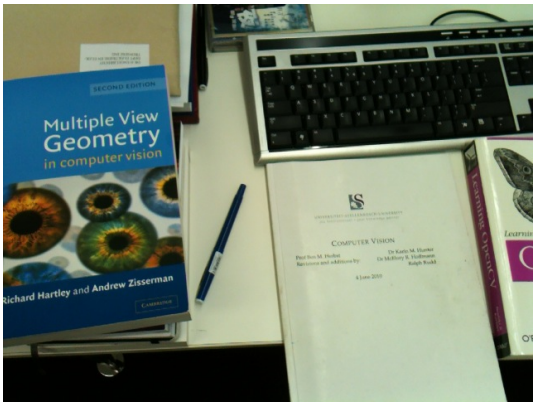
	Sample1	**Sample2**	**Sample3**	**Sample4**	**Sample5**	**Sample 6**
time	67	88	32	150	141	101
numcorners	21	40	10	67	60	47

Pose Estimation Accuracy:

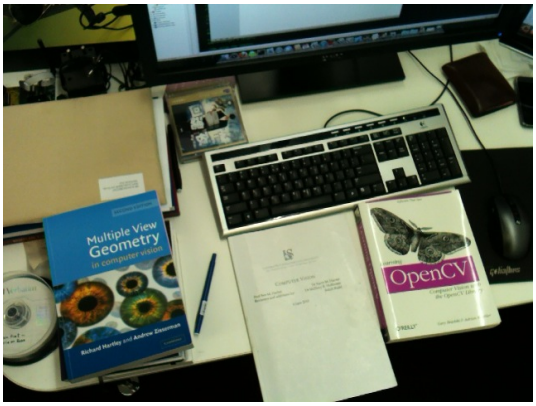
X-axis		**Y-axis**		**Scale**	
Movement (px)	**Error (px)**	**Movement (px)**	**Error (px)**	**Actual**	**Measured**
0	0	80	2	2	1.5
20	1	-100	9	4	3.7
150	5	110	10	8	7.06
-170	8	-120	5	0.5	0.48
-10	1	140	3	0.25	0.28
120	6	-140	15	0.125	0.18
-70	3	30	0		
50	3	-30	2		
100	4				
10	1				
-100	5				

Rotation	
Degrees	**Error (px)**
~15	1
~30	8
~45	11
~60	32
~75	76

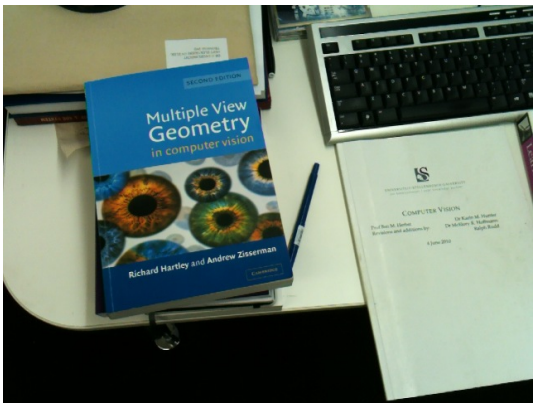
Appendix B



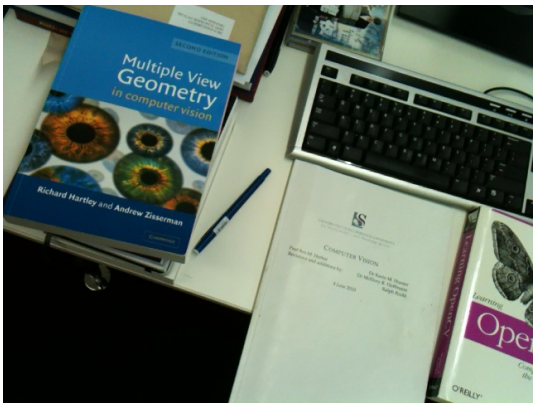
(a) Original Image



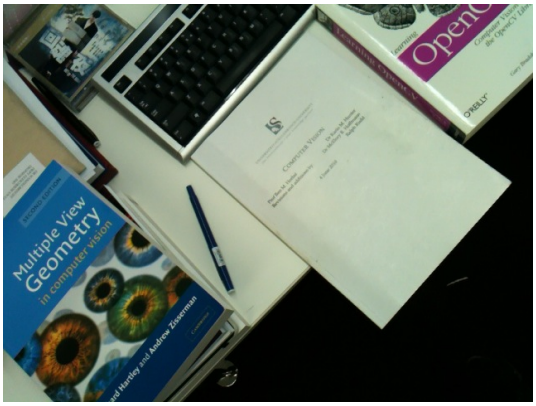
(b) Scaled Image (x 1.5)



(c) Translated Image (7 cm)



(d) Rotated Image (± 15 degrees clockwise)

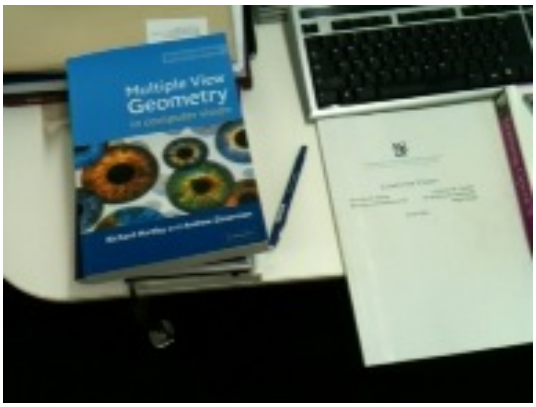


(e) Rotated Image (± 50 degrees counter-clockwise)

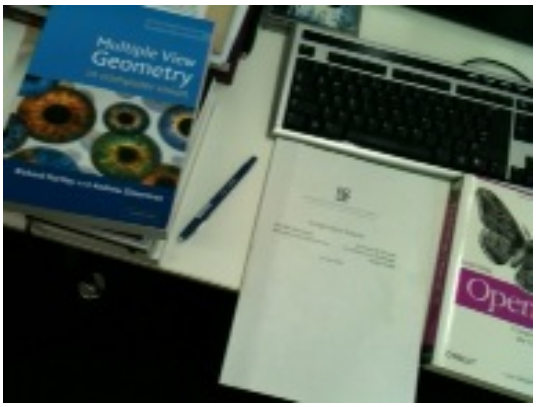
Figure 1: *Appendix B: High Quality images used for Corner detection, SURF, Optical Flow and System tests*

(a) Original Image

(b) Scaled Image (x 1.5)



(c) Translated Image (7 cm)



(d) Rotated Image (± 15 degrees clockwise)

(e) Rotated Image (± 50 degrees counter-clockwise)

Figure 2: *Appendix B: Low quality images used for Corner detection, SURF, Optical Flow and System tests*